WORKLOAD-BASED AUTOMATED INTERFACE MODE SELECTION

THESIS

Andrew J. Compton, Captain, USAF

AFIT/GCE/ENG/12-03

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright Patterson Air Force Base, Ohio

AFIT/GCE/ENG/12-03

WORKLOAD-BASED AUTOMATED INTERFACE MODE SELECTION

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Engineering

Andrew J. Compton, B.S.

Captain, USAF

March, 2012

AFIT/GCE/ENG/12-03

WORKLOAD-BASED AUTOMATED INTERFACE MODE SELECTION

Andrew J. Compton, B.S.
Captain, USAF

Approved:

/signed/

Dr. Gilbert L. Peterson (Chairman)	Date

/signed/

Dr. Paul R. Havig (Member)	Date

/signed/

Dr. Michael E. Miller (Member)	Date

Abstract

The increase in the size of the Air Force's Unmanned Aerial Vehicle (UAV) fleet, and the desire to reduce operational manning requirements, has led to an interest in Multiple Aircraft Control (MAC) technology. The MAC concept is highly prone to operator overload, as each aircraft could be engaged in unpredictable dynamic scenarios, inducing severe workload levels on the operator. Attempts to alleviate the operator's workload through the use of interface automation has not been entirely successful. While automation is not susceptible to the same cognitive limits as a human operator, it can be brittle. To attempt to mitigate the potential of operator overload, while reducing the risk of automation failures, this research introduces an agent into the system interface to assume responsibility for managing automation mode selection. The agent uses a novel dynamic scheme for determining how and when to introduce automation assistance to the operator. The agent employs reinforcement learning to learn and adapt to each individual operator's unique abilities and also adapt to operator proficiency changes over time through the use of online learning. By automating tasks at appropriate times, the agent helps the system balance the operator's workload level, striking the best possible balance between operator awareness and overall performance, while reducing the potential for operator overload. This concept was tested through the use of human trials with the Workload and Automation Level Response Simulator (WALRuS) testbed and automated experiments with cognitively modeled agents. Testing established a correlation between workload and performance for each subject, and each subject's surrogate cognitive agent showed that the interface agent could successfully learn the human's performance profile and increase their potential performance.

Acknowledgements

First and foremost I offer my sincerest gratitude to my original research advisor, Lieutenant Colonel Brett Borghetti, who helped me focus my ideas into a thoughtful and Air Force relevant thesis topic. His leadership and encouragement helped to get me off to a good start with a solid foundation for my topic and the production of thesis deliverables.

As sometimes happens, Lieutenant Colonel Borghetti received deployment orders and Major Kennard Laviers stepped in as my new advisor. Major Laviers has my eternal gratitude for his friendly hands-on approach, genuine interest and support of my topic, and constant encouragement.

Soon after taking me on as an advisee, Major Laviers also received deployment orders. I, as it turned out, was perhaps a bad luck charm. I would like to thank Dr. Gilbert Peterson for helping to shape my education as the instructor for my introductory artificial intelligence courses and for taking on the role of my third and final thesis advisor.

I would also like to thank my first supervisor of three years, Dr. Paul Havig, at the Warfighter Interface Division's Battlespace Visualization Branch. As my supervisor and as a member of my thesis committee, Dr. Havig was a constant source of ideas, encouragement, and enthusiasm.

The experiment used to test my agent was modeled after an experiment developed by researchers Eric Geiselman and Eric Heft at the Battlespace Visualization Branch. Both have my sincerest thanks for their assistance.

I would like to thank Dr. Micheal Miller for stepping in to round out my thesis committee in response to a last minute request.

Andrew J. Compton

Table of Contents

Page

List of Figures

List of Tables

List of Algorithms

WORKLOAD-BASED AUTOMATED INTERFACE MODE SELECTION

1. Introduction

Imagine a scenario where a single operator is given responsibility for monitoring and controlling several unmanned aircraft. For each aircraft, the operator must maintain a high level of situational awareness in order to accomplish the respective missions of the aircraft. Additionally, the operator must be able to spot unexpected occurrences and recognize potential failure modes before they degrade mission performance. In a perfect scenario, with no failures or unexpected occurrences, the workload placed on the operator might be manageable and the operator may be able to perform well. However, what happens when the operator's workload is increased by a dynamic mission tasking, or the occurrence of a failure? In such a case, the increase in workload can quickly overload the operator, leading to decreased performance, loss of military assets, and potential disaster for the mission.

1.1 Research Area

Designers of system interfaces are faced with many challenges. One of the largest challenges involves the trade off between how much control and information should be exposed to the operator. Exposing less control and information has the benefit of offering a simpler, easier to use interface, but risks limiting the operator's ability to perform complex actions. Exposing more control and information gives the operator the ability to understand the state of the system better and take more complex actions, but at the expense of the systems ease of use and overall simplicity.

Additional considerations interface designers must contend with are the needs of their intended user base. Different users have different skill levels. Even with equal training and experience, the performance levels of two operators may be vastly different under similar conditions [13]. Traditionally, interface designers have dealt with this hurdle in one of two ways. The first is by making the interface static [14, 13]. The static interface cannot be changed, and is normally targeted at the average user's ability. The second is by making the interface adaptable [13, 6]. Adaptable interfaces are those that allow the operator to

Figure 1 Vigilant Spirit UAV Control Console [1].

change the interface by adding, moving, and removing elements to customize it to their preference.

Recent research and development efforts have focused on creating adaptive interfaces [13, 3, 5, 27]. Adaptive interfaces automatically reconfigure themselves to improve the efficiency of the operator's interactions with the interface. Interface changes are enacted through the use of an Artificial Intelligence (AI) software agent. Software agents are actors that sense their environment, make decisions, and then take actions [35]. Our agent is a rational actor that monitors the behavior and performance of the operator, determines when and how interface adaptations would have the potential to improve performance, and then enacts those adaptations.

1.2 Multi-Tasking Environments

The impetus for this research comes from the difficulties encountered by designers working to build systems to enable Multiple Aircraft Control (MAC) [8]. MAC is a relatively new concept relating to the UAV control domain. The current generation of UAV control systems are setup as remote cockpits where each operator pilots a single UAV. MAC seeks to increase efficiency and reduce manning requirements by allowing one operator to manage multiple aircraft. The Vigilant Spirit control console [1], shown in Figure 1, is one of the leading testbeds for developing MAC systems.

Several problems currently hinder the development of MAC technology. The most significant problem involves operator workload. Managing a single aircraft can be a work-

load intensive task, but it remains a task within the cognitive abilities of most operators. Managing multiple aircraft introduces the chance of workload intensive scenarios overlapping, leading to operator overload. When overload occurs, performance decreases, situational awareness is hindered, and the chance of catastrophic occurrences increases. One of the major contributing factors during a period of overload is the varied types of tasks the operator is required to perform [2, 47, 24].

UAV operators are required to perform many different types of tasks. For a single UAV, the operator must manage expendable resources such as fuel and armament, the route of the UAV, communications, and monitor overall system status. With multiple aircraft, the operator must be able to correctly perform several instances of each of these tasks simultaneously. For the purposes of this research, the interface enhancements that are being examined will be tested using a series of tasks from the Multiple Attribute Task Battery (MATB) [26]. The MATB, further explained in Section 4.3, contains tasks that are representative of those which are encountered by pilots and UAV operators. These include system monitoring, tracking, communications, and resource management.

1.3 Research Objectives

The goal of this research is to examine the implications of reducing operator workload by offloading unmanageable portions of the operator's workload to automation. Maintaining the operator's situational awareness is a key objective, and therefore one of the system's primary goals is to keep automation activities at the lowest possible levels needed to maintain the operator's performance. Operator overload is one of the principle concerns with MAC technology, and interface automation is thought by many to be the best way to mitigate it. However, past efforts to introduce automation to interfaces have encountered issues with operator awareness and dependence. While increased automation may lead to higher performance, it can also lead to a loss of operator awareness and an increased dependence on the automation by the operator. In addition, systems that employ a 'one size fits all' automation function allocation scheme will likely never be optimal for all operators. The experience of each operator, their background, and their innate skills all play a role in determining how well they will respond to different workload conditions. The overall

hypothesis for this research is as follows: Significant performance increases can be achieved and operator overload conditions can be mitigated through the introduction of personalized automation assistance. Agents that help the operator by introducing automation into the interface during periods of high workload should allow the overall system to achieve a higher level of performance than would otherwise be achieved without automation. To test this hypothesis, an adaptive agent was developed and empirically evaluated. The results obtained while using the adaptive agent are compared to results obtained without the use of the agent.

1.4 Methodology

The workload placed on an individual operator can be reduced in two ways. The first is by designing interfaces in an intelligent and intuitive manner so tasks are completed effectively with the least amount of cognitive load on the operator necessary to complete the task [14]. Ideally, all good interface designs should strive to meet this objective. The second is by offloading some of the operator's workload onto other human operators or onto automation [16]. Two questions arise with regard to offloading portions of the workload. The first is the manner in which the workload should be divided, and which portions of it should be offloaded. The second is the timing, or when the workload should be offloaded and when it should be re-assumed by the original operator.

Our system handles the problem of how and when to introduce automation assistance through a reinforcement learning approach. This approach requires each operator to train the agents responsible for controlling the interface. Once trained, the agents are personalized to specific operators. The agent is then responsible for maintaining the operator's performance at a predefined level. When workload increases, and a performance decrease is predicted, the agent begins to automate activities that it has determined will increase overall performance. The agent therefore provides only the lowest level of automation assistance required to maintain the desired performance, and minimizing automation dependence by the operator.

1.5 Assumptions and Limitations

This approach relies on several assumptions about the manner in which operators are able to handle workload. The basic premise of this research assumes that humans do in fact have limited cognitive abilities [22]. The maximum workload threshold for each operator may vary, however each operator has a limited level of workload that they can effectively manage. Additionally, we assume there is a correlation between workload and performance. While the exact relationship may be somewhat complex, in general we expect that as workload increases, especially as it surpasses the operator's cognitive limit, the operator's performance will decrease.

A limitation for our system is the time spent by the operator training the agents. A newly initialized agent, not yet trained or pre-populated with training data, is unable to assist the operator. Only after the agent's policy converges will it be able to accurately provide automation assistance.

1.6 Implications

The AMS agent concept was tested through the use of human trials and automated experiments with cognitively modeled agents. Testing established a correlation between workload and performance for each subject, and each subject's surrogate cognitive agent showed that the interface agent could successfully learn the human's performance profile and increase their potential performance.

This approach poses new challenges that may affect system operators. One of these challenges is mode confusion [7]. It may be possible for the operator to become confused as to the current state of automation in the system. Another issue is dependence or over reliance on the system's automation activities [10]. The operator may grow comfortable with a lower level of performance than they are capable of if they know the system will step in and make up for their inaction or inattentiveness.

The system has the potential for very positive implications with regard to multi-tasking command and control systems. If the approach succeeds in increasing performance and mitigating the negative effects of high workload scenarios, it would allow fewer oper-

ators to manage greater numbers of aircraft in MAC environments and pave the way for other environments to do the same.

1.7 Summary

The AMS agent concept is designed to improve performance for MAC technology. This is accomplished by introducing automation into operator interfaces according to a schedule determined through Reinforcement Learning (RL). This approach allows the agent to adapt to the different skill and experience levels of each operator, and allows the agent to continue to adapt as those skill and experience levels change. If successful, the AMS agent concept has the potential to mitigate operator overload conditions, and make MAC operations more productive and successful for the United States Air Force (USAF).

This thesis documents the research, development, and testing required to develop the AMS concept. To evaluate related research, a literature review was conducted on varying interface automation related topics. This literature review is documented in Chapter 2. Design and implementation considerations for the AMS agent are documented in Chapter 3. The development and implementation of the WALRuS testbed is documented in Chapter 4. Results obtained from testing is presented in Chapter 5, followed by a conclusion and recommendations in Chapter 6.

2. Literature Review

This thesis presents a method by which Reinforcement Learning (RL) is integrated into a user interface to improve a systems overall performance by minimizing the effects of operator overload. This research focuses on human interfaces for real-time command and control environments, such as those found in Unmanned Aerial Vehicle (UAV) control systems. These environments can be task intensive, and place heavy workloads on system operators. When the operator workload exceeds their capability, the operator becomes overloaded and the system's performance suffers. The goal of integrating AI into the interface is to provide assistance to the operator that can help mitigate the negative effects of operator overload and help reduce the possibility of an overload occurring.

Before analyzing different interface design mechanisms, this literature review examines the limits of human cognition. Intelligent assistance can be provided to the operator of a system in many different ways. This literature review seeks to classify the different types of intelligent assistance and explore the methods by which they operate, the challenges they pose to end users and system designers, and examine different examples of system implementations. This review also identifies general goals and tenets of good intelligent system designs as well as good experimental designs for the establishment and analysis of metrics to accurately determine the effectiveness of any such system.

2.1 Limits of Human Cognition

It has been shown that the average human brain has a limited working memory [22]. In 1956, G.A. Miller conducted experiments that demonstrated this limit. Miller's experiments revealed that the number of items a human could maintain in their working memory was in the range of seven plus or minus two. These findings established the fact that there are indeed finite limits to human cognition, and laid the foundation for cognitive load theory. The implication of this cognitive limitation is profound. With only a limited working memory, humans are unable to effectively manage large numbers of tasks simultaneously [22]. In scenarios where it is desirable for a single human to successfully

exceed this limitation, methods of augmenting the human's abilities to reduce workload must be implemented.

2.2 Types of Interface Assistance

The goal of an automated assistant is to maximize the potential for positive benefits achieved through its use while minimizing the potential for negative consequences it may have on short or long-term performance [17]. To that end, different types and levels of automated assistance can be provided to the user of a system. Types of assistance can be varied and complimentary. Shortcomings in one method may be mitigated by strengths in another. The level of assistance a system provides can also be variable based on user experience or preference.

This literature review divides methods of automated assistance into two categories. The first category involves using dynamic layouts, or configurations that change or can be changed. These include both adaptive [17] and adaptable interfaces [16]. The second category includes interfaces that reduce operator workload by offloading portions of the workload to automation.

2.2.1 Dynamic Interfaces.

Dynamic interfaces can adapt to better suit the needs of an individual user, type of user, particular task, or operating mode [17]. Adaptation is achieved by allowing the operator to customize the interface, or by employing a system that automatically modifies the interface in an attempt to improve the operator's performance. In this case, both methods of adaptation do not necessarily have to be mutually exclusive.

It is important to note the distinction between adaptive interfaces and adaptable interfaces. Interfaces that are adaptable are those that allow the user to change them. Interfaces that are adaptive have a built-in mechanism that allows them to change automatically [20].

2.2.2 Adaptable and Adaptive Interfaces.

Many of the elements in modern computer software are adaptable. Programs such as Microsoft Word offer many customization options to the end-user. This includes the ability to add or remove menus and tool bars,

add items to tool bars, remove items from tool bars, and the ability to display material in a variety of sizes and formats. Adaptive interfaces are less common. Older versions of Microsoft Office had a setting which would allow adaptive drop-down menus. The most commonly used items would automatically be placed at the top of the list and the least commonly used items would migrate to a fly-out menu. Microsoft Office is not an environment that requires quick decisions nor does it have heavy workloads however. Games and simulations do often impose heavy workloads and require fast response, and may be a better target for these types of interface features. Many computer games already use adaptive interfaces to present only the most relevant information to players. In most First Person Shooter (FPS) games [11, 42, 4], when a player climbs into a vehicle, information about their avatar's current status becomes irrelevant and is replaced on the display with information about the vehicle they have climbed into.

2.3 Interface Automation

Automating elements of a system interface is another method by which workload reduction can be achieved [38, 9, 12, 34, 8, 27, 15]. The premise behind this idea is that automation can take responsibility for a task, or series of tasks, freeing up cognitive obligations that would otherwise have been the responsibility of the operator. Aspects relevant to workload reduction through automation include taxonomies for classifying automation, automation cycle times, and triggering methods used to determine when automation modes should be altered.

2.3.1 Levels of Automation. Levels of Automation (LOA) provides a taxonomy that has proven to be a successful means of ranking automation assistance [44]. Endsley and Kaber [12] propose a ten-level taxonomy that ranks a system's LOA, shown in Table 1. This taxonomy divides the gap that exists between manual control and full automation into distinctly identifiable categories. The taxonomy delegates four distinct tasks: monitoring, generating, selecting, and implementing. Monitoring is simply the task of perceiving the state of the activity or system status. Generating is the task of formulating actions that will maximize utility or achieve a goal. Selecting is the task of choosing which action will

be executed. Implementing is the task of executing the selected action. Each of taxonomy's ten levels delegate these four tasks to either the human operator or the computer. At level one all four tasks are delegated to the human operator. At level ten all four tasks are delegated to the computer. The levels in between use different combinations, rank ordered by increasing level of computer control.

Table 1 Hierarchy of levels of automation applicable to dynamic-cognitive and psychomotor control task performance [12].

	Roles			
Level of automation	Monitoring	Generating	Selecting	Implementing
(1) Manual control (MC)	Human	Human	Human	Human
(2) Action support (AS)	Human/Computer	Human	Human	Human/Computer
(3) Batch processing (BP)	Human/Computer	Human	Human	Computer
(4) Shared control (SHC)	Human/Computer	Human/Computer	Human	Human/Computer
(5) Decision support (DS)	Human/Computer	Human/Computer	Human	Computer
(6) Blended decision making (BDM)	Human/Computer	Human/Computer	Human/Computer	Computer
(7) Rigid system (RS)	Human/Computer	Computer	Human	Computer
(8) Automated decision making (ADM)	Human/Computer	Human/Computer	Computer	Computer
(9) Supervisory control (SC)	Human/Computer	Computer	Computer	Computer
(10) Full automation (FA)	Computer	Computer	Computer	Computer

In their paper, Endsley and Kaber [12] describe an experiment devised to measure system performance at each level in the taxonomy. The experiment consisted of a vigilance task implemented in the Multitask framework [45]. Vigilance tasks are trials in which the subject's ability to detect and respond to psychomotor stimuli is tested. Subjects in the experiment were shown shapes that moved toward the center of a polar grid. The task of the operator was to collapse the shape by clicking on it repeatedly before it could expire or collide with another shape. Each shape had values displaying how much time was left before it expired, how many points could be earned if it was collapsed, and how many penalty points would be incurred if it expired or collided. The experiment was executed at each of the ten levels of automation. The experiment recorded the number of total points achieved as well as metrics designed to show levels of operator SA and operator-perceived workload. When plotted, the results were mixed with no clear winner for all categories. Higher levels of automation had lower operator-perceived workloads, but higher recovery times when artificial system errors were introduced. Lower levels of automation had higher operator-perceived workloads, and lower recovery times for system errors. The overall score that was awarded was maximized during trials with level two (action support) and level three (batch processing) assistance.

Calculating an overall system performance score for these types of trials sums the points earned minus the sum of the penalties incurred. In order to measure the perceived response driven operator workload, the system would occasionally halt the experiment and query the user about the state of objects in the experiment. If the operator could correctly answer questions then they must have good situation awareness and are not likely to be overloaded. If they are unable to answer the questions then it is likely that they are being overloaded and are not fully able to absorb all of the details of the current system state. The measured response time is the time it takes for the operator to realize the system has failed and take steps to institute manual control. For the purposes of Endsley and Kaber's experiment, a dialog was displayed informing the operator that the system had failed. At level one, manual control, there is nothing to recover from because the system does not perform any tasks for the operator. At level two, action support, the operator is still actively working the experiment at a low level and an automation failure is unlikely to have a large effect. At other levels in the taxonomy the operator's attention is focused on higher-level planning and monitoring. Automation failures at these levels have a better chance of catching the operator off guard, with the shock leading to increased response times. It is likely that the operator's dependence on the system could also be inferred from the response time metric.

The LOA taxonomy Endsley and Kaber present is well regarded in the HCI community and appears in many human factors publications [36, 39, 29, 19]. The taxonomy levels, as defined in the table above, are suitable definitions for automation levels in future work. There are several issues with the human trial that may prevent the results of the study from being universally applicable [12]. The first issue comes from the fact that there is no clear winner when examining the different scoring systems used. The level of automation used for a particular task might be completely dependent on which of the three metrics measured is most important to maximize (or minimize). Another issue lies with the type of activity that is being automated. This study examined the metrics for only one type of activity. It may be the case that different activities can be optimized with different levels of automation. A second potential flaw in the study involves the types of participants [12]. The study used graduate students as subjects. Each subject was given

the same training routine before executing the tasks. Different levels of operator skill may require different levels of automation to maximize performance. The largest flaw in the experiment may be the way the author coded the decision-making algorithm. The author states that the algorithm could have been capable of picking perfect outcomes to maximize points every time, but chose not to in order to introduce realism. The impossibility of a perfect algorithm may be unrealistic. Algorithms for many complex decision-making tasks could be flawless, or nearly so. It is likely that the results of the experiment would shift in a non-linear fashion if the skill-level of the decision algorithm were changed.

The task that the system was assigned to perform in Endsley and Kaber's experiment was extremely simple. The task was little more than simple planning and reflex response. Long-term planning was limited to the few seconds that any series of boxes would be moving across the screen. An experiment using this LOA taxonomy applied to a MAC control system would be vastly different. Dynamic scenarios with unmanned aircraft require efficient long-term planning as well as the flexibility to adapt in the face of unexpected circumstances. Endsley and Kaber's experiment was designed to determine the effectiveness of a system operating at the different levels defined by their LOA taxonomy. Because they were testing the taxonomy and not necessarily the system itself, they programmed an automation agent that was incapable of being perfectly efficient for a task where perfection was attainable. This gap left potential room for the operator in manual mode to exceed the performance of the system in a fully autonomous mode. This gap would be required to effectively differentiate the results if they were equal in manual and autonomous modes. The likelihood of having a perfect autonomous assistant for the MAC domain is low. Because of this, there is no need to handicap the system as Endsley and Kaber have done.

2.4 Automation Cycle Times

The selection of cycle times for manual control and automation periods has been shown to be an important consideration that affects overall system performance [38]. Longer automation cycle times may cause operators to be less vigilant. This complacency has the potential to decrease performance during automation periods and may be

especially problematic during transitions from automated modes to manual control. Several automation studies have examined the affects of medium to long duration automation cycles [37, 9]. Because unmanned aircraft often have missions that require only brief periods (as short as 30 seconds) spent performing the critical portions of their mission, there is a need to examine the effects of manual control and automation cycles of less than one minute. In the experiment conducted by Scallen [38], cycle times for 15, 30, and 60 seconds were examined. The results showed that shorter cycle times produced better performance, while longer cycle times induced less mental load on the subjects.

2.5 Automation Triggers for Adaptive Aiding

Determining when to engage automation or dynamic elements within the interface is a primary component for an automated assistant [38]. Scallen describes and analyzes several potential automation triggers. The first trigger relies on data from the operating environment, such as the status of a vehicle or conditions outside the vehicle. The second utilizes the operator's performance to trigger automation. The third relies on physiometric data recorded from sensors attached to the system's operator.

2.5.1 Environmental Conditions. Under this method, elements of the operational environment must be specifically identified as automation triggers [37]. Scallen states that this method may be practically limited, as the creators of the system would have to anticipate all potential conditions that could present themselves to the system. For environments where it is unlikely that the designers would be able to predict all possible conditions, a system using such a trigger could become disrupted by unexpected input.

An experiment conducted by Bennet [3] attempted to measure performance differentials for an adaptive interface that used an aircraft's location as an environmental trigger. When the aircraft strayed from its mission path, the display would change to include information from an artificial flight director. Additionally, a force-feedback mechanism in the control system was implemented where by the friction in the control stick would increase as the aircraft strayed from its predetermined path. Bennet's experiment was conducted

using eight US Air Force Pilots. The results of the experiment, while not statistically significant, did show a likely performance boost for low-level navigation activities.

2.5.2 Operator Performance. The goal of the adaptive system is to improve operator performance. Operator performance therefore makes an appealing metric for triggering automation assistance. In an analysis of triggering mechanisms for adaptive function allocation, Scallen [37] points out a flaw in this reasoning. For the aircraft domain, measuring performance can be a difficult task. Performance is a measure of the pilot's efficiency and progress toward meeting mission objectives. The accuracy of the performance metric would depend on how well the system can predict the outcomes of the actions that the pilot is taking.

Another approach to using operator performance as a triggering mechanism relies on the correlation between performance and workload. Wickens [47] describes the relationship between performance and workload as being complex. It is not always the case that increases in workload will lead to decreases in performance. Just as having too much workload can lead to overload and performance decreases, having too little workload can lead to boredom which will also lead to decreased performance [24].

Wickens' Multiple Resource Theory (MRT), shown in Figure 2, allows cognitive tasks to be decomposed and analyzed based on the individual components used in the taxonomy. Using MRT, a careful analysis of tasking can be performed to determine which tasks are complementary, and which tasks will exhibit resource collisions. These collisions can be reasonably expected to create overloads, thereby negatively impacting performance.

Calhoun [9] made use of performance-based triggers to conduct an Adaptive Level of Automation (ALOA) study in a simulated multi-UAV control task. This experiment required subjects to monitor several UAVs, manage flight paths and alerts, and analyze sensor imagery returned by the simulated aircraft. The system made use of a performance-based trigger to implement multiple levels of automation for aiding with the image analysis task. The results of the study showed that an improvement in operator response times and a reduction in subjective workload was achieved through the use of the adaptive interface when compared to results where the adaptive interface was not used.

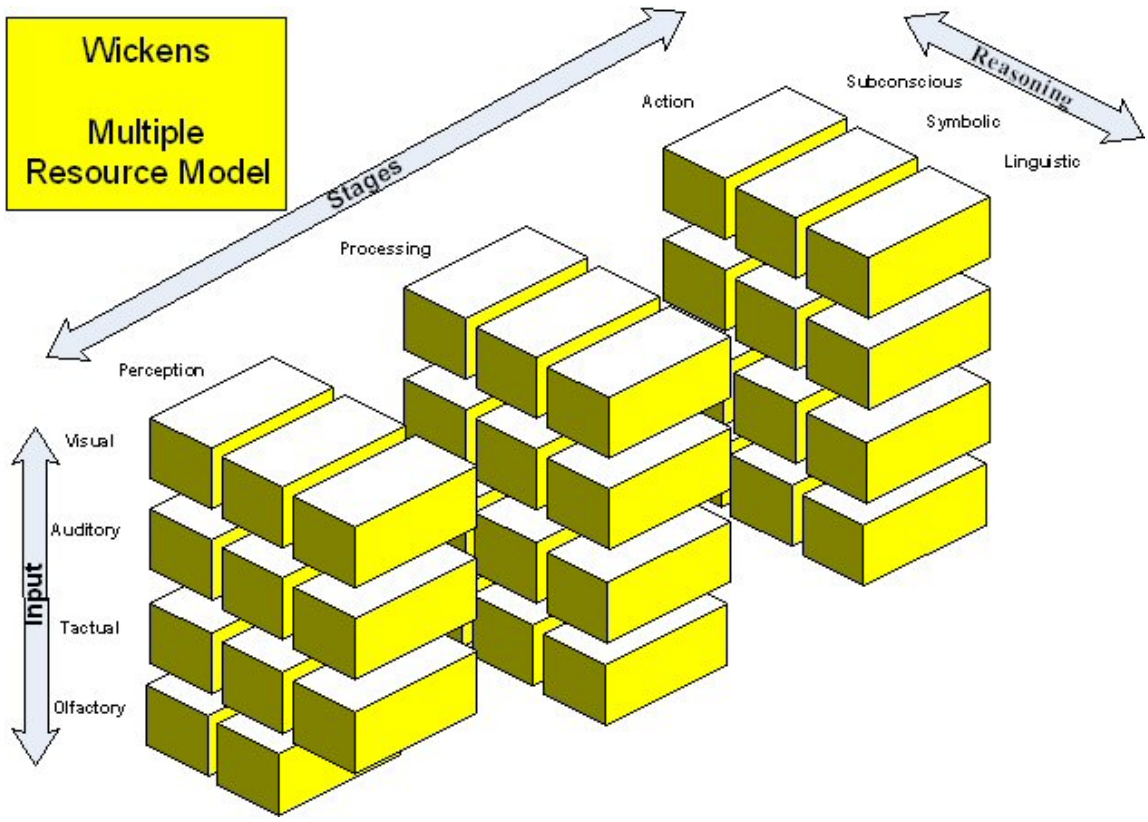


Figure 2 Wickens' Multiple Resource Theory (MRT) Model [41].

2.5.3 Physiometric Indicators. Physiological reactions occur in humans during times of stress [48]. Stress has been shown to correlate well with mental workload [40, 48]. Because physiological indicators are always present, they offer good potential sources for measuring a persons mental workload [48]. This physiologically derived mental workload value can then be used as a trigger for enabling and disabling interface automation, forming a closed-loop system to regulate workload. Prinzel [31] conducted a study to test the effectiveness of such a system. Their system made use of an Electroencephalogram (EEG) to measure mental activity. The study showed that there was potential for the success of such a system based on the EEG index values recorded in their experimental data. Regardless of exactly which physiological indicators are used, it is imperative that the system be capable of accurately determining the operator's state [48].

2.6 Usability Challenges

The primary goal of the methods examined in this research is to increase operator-system performance. There are however certain challenges that can prevent the help such a system provides from having its intended effect. These challenges are one of the main reasons for the existence of the human factors discipline. Factors such as transparency, predictability, trust, and over reliance can play a large role in determining both the short and long-term success and effectiveness of any system that involves Human Computer Interfaces (HCI) [17]. If these factors aren't carefully considered, and their related issues mitigated, the resulting system may have usability problems that outweigh the benefits the system is intended to provide [18, 21].

For a system's operator to maintain the best possible level of awareness they must have a good understanding of the motivations that drive all decision making processes [17]. When decision making takes place entirely in the head of the operator, his or her own motivating factors are obvious and internally available. However, when decision-making tasks are outsourced to automation, the operator may become disconnected from or entirely unaware of the considerations that are taken into account in order to arrive at a course of action. This lack of situational awareness only leads to further issues with predictability, trust, and over reliance.

Predictability is one of the most important elements of a decision support system [17]. The operator of such a system should always be able to determine with a significant amount of certainty how the system will behave for a given state and input. When the system behaves spontaneously it can leave the operator in a state of confusion and endanger the overall goals of unobtrusiveness and controllability [18]. In order to maintain both predictability and transparency the user must have access to a system's rationale, including why the system is making recommendations and how it is generating them [6].

A system must engender a certain level of trust with its operator if it is to be useful [28]. If an operator has no trust in the system then they will not derive the potential benefit it has to offer. Conversely, if the operator trusts the system's advice excessively they will become over reliant on the system [18]. The best balance is achieved when the operator has a good level of awareness with regard to the system's decision-making processes and is fully competent in the field for which the decisions are being made.

The use of decision support systems may have negative long-term side-effects such as over reliance [10]. If the operator chooses to rely too heavily on decision-making assistance, their own ability to make unaided decisions could become compromised. In cases where the system providing decision-making support fails, the operator's performance may actually be worse than it otherwise would have been in manual mode, due to being out of practice.

2.7 General Goals and Tenets

The prior work includes lists of general goals and tenets that establish basic principles governing the way a proposed system should operate. Establishing these principles early in the beginning of the design process is important because they provide the foundation for the proposed system.

One system employing a mixed-initiative interface approach summed up its usability goals as follows: maintain user control, provide customization support non-intrusively, and maintain predictability and transparency [6]. Others begin the design process with more defined and elaborate goals and goal descriptions such as those listed in Table 2.

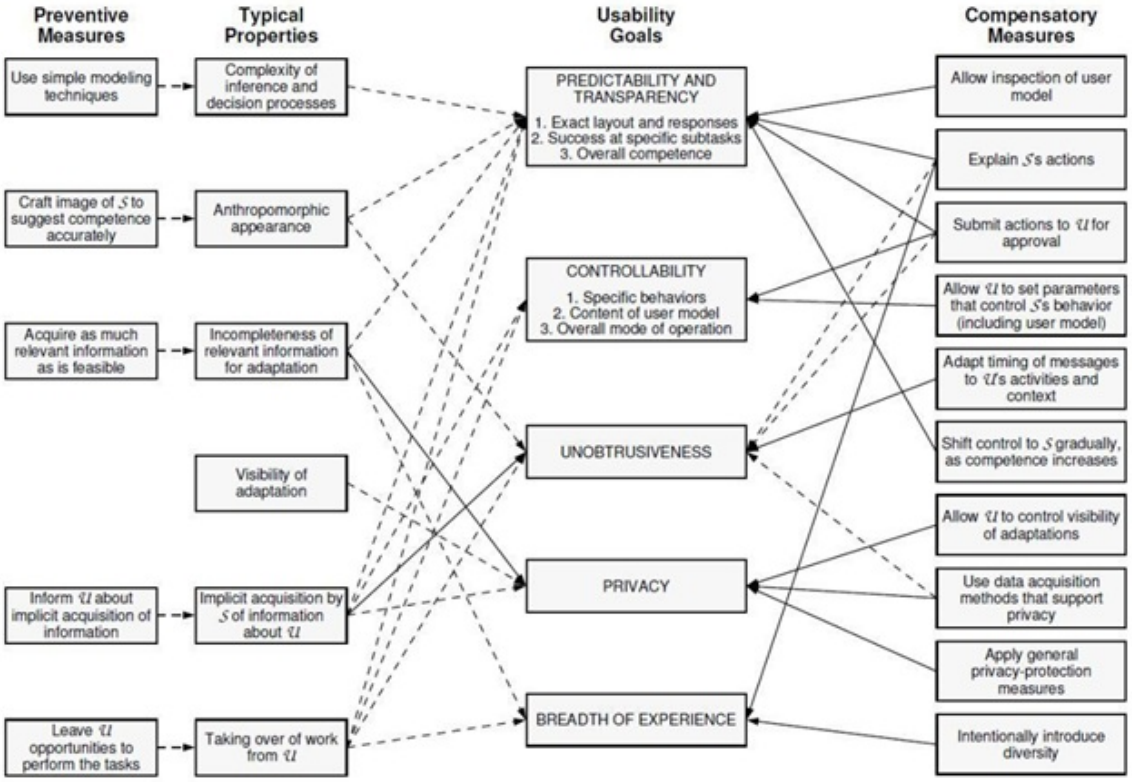


Figure 3 Overview of usability challenges for user-adaptive systems [18].

Table 2 Critical factors for the effective integration of automated services with direct manipulation interfaces [16].

1	**Developing significant value-added automation.**	It is important to provide automated services that provide genuine value over solutions attainable with direct manipulation.
2	**Considering uncertainty about a user's goals.**	Computers are often uncertain about the goals and current focus of attention of a user. In many cases, systems can benefit by employing machinery for inferring and exploiting the uncertainty about a user's intentions and focus.
3	**Considering the status of a user's attention in the timing of services.**	The nature and timing of automated services and alerts can be a critical factor in the costs and benefits of actions. Agents should employ models of the attention of users and consider the costs and benefits of deferring action to a time when action will be less distracting.
4	**Inferring ideal action in light of costs, benefits, and uncertainties.**	Automated actions taken under uncertainty in a user's goals and attention are associated with context-dependent costs and benefits. The value of automated services can be enhanced by guiding their invocation with a consideration of the expected value of taking actions.
5	**Employing dialog to resolve key uncertainties.**	If a system is uncertain about a user's intentions, it should be able to engage in an efficient dialog with the user, considering the costs of potentially bothering a user needlessly.
6	**Allowing efficient direct invocation and termination.**	A system operating under uncertainty will sometimes make poor decisions about invoking--or not invoking--an automated service. The value of agents providing automated services can be enhanced by providing efficient means by which users can directly invoke or terminate the automated services.
7	**Minimizing the cost of poor guesses about action and timing.**	Designs for services and alerts should be undertaken with an eye to minimizing the cost of poor guesses, including appropriate timing out and natural gestures for rejecting attempts at service.
8	**Scoping precision of service to math uncertainty, variation in goals.**	We can enhance the value of automation by giving agents the ability to gracefully degrade the precision of service to match current uncertainty. A preference for "doing less" but doing it correctly under uncertainty can provide users with a valuable advance towards a solution and minimize the need for costly undoing or backtracking.
9	**Providing mechanisms for efficient agent-user collaboration to refine results.**	We should design agents with the assumption that users may often wish to complete or refine an analysis provided by an agent.
10	**Employing socially appropriate behaviors for agent-user interaction.**	An agent should be endowed with tasteful default behaviors and courtesies that match social expectations for a benevolent assistant.
11	**Maintaining working memory of recent interactions.**	Systems should maintain a memory of recent interactions with users and provide mechanisms that allow users to make efficient and natural references to objects and services included in "shared" short-term experiences.
12	**Continuing to learn by observing.**	Automated services should be endowed with the ability to continue to become better at working with users by continuing to learn about a user's goals and needs.

2.8 Gaps in Research

Two omissions from the studies examined are valid trials establishing solid metrics for the importance that trust and over-reliance can have [46]. Most of the articles at least mentioned trust and over reliance as factors, but there were no thorough attempts to quantify either factors through human trials in a study. In a mixed-initiative semi-autonomous system the operator will likely have to have some level of trust in the system before allowing it to perform certain tasks. Trust in automation is something that should be further examined when deciding if such a system is feasible and if it could be effective. Over-reliance by the operator was somewhat touched upon in Endsley and Kaber's study [12]. They measured the time it took for a human to recover from a system failure. While one may be able to infer operator reliance from this metric, it is not exactly the same thing. If the system is constantly performing tasks that were once performed by the operator,

his skill may begin to atrophy and reduced system performance will be the likely result when the automation fails. This work addresses the over-reliance issue by maintaining automation at the lowest levels needed to produce a certain level of performance.

2.9 Summary

The literature survey examined several different methods for integrating automation into a user interface. Methods such as interface modification in the Dynamic Adaptive Interface (DAI) and Intelligent Adaptive Interface (IAI) concepts are more covert. Rather than taking on operator tasks the intelligence works to improve operator performance by optimizing the flow of information to and from the operator. The Adaptive Intelligent Agent (AIA) portion of the IAI concept and the LOA taxonomy and study presented by Endsley and Kaber offer a different style of intelligence integration through the use of mixed-initiative agents. This style is more overt. After the first level of autonomy, manual mode, each successive level allows the system to either make suggestions or to take action. In this case the goal of the system is to optimize overall performance.

Each of these methods of automation integration have merit and could use further study. All three methods are based on sound engineering principles and include human factors considerations. Some of the features of each overlap, and the three methods listed here may not be mutually exclusive. The idea of an Intelligent Adaptive Interface that makes use of Adaptive Intelligent Agents that each operate at independent Levels of Automation in order to maximize the overall performance of the human-machine system is particularly intriguing. Each agent in such a system could work to maximize a given goal. These agents could be managed by the operator through the interface or by an additional control agent.

3. Agent Development

Operators of future command and control systems will be challenged by ever increasing workloads. Complex interfaces that require operators to monitor and perform multiple distinct tasks ultimately lead to operator overload. Building automation into Human Computer Interfaces (HCI) has shown promise as a method for mitigating such overloads [12, 37, 9, 34]. Determining exactly when and how to provide automation assistance during periods of high workload has become a key factor for successful systems. The presented approach involves implementing automation through an Intelligent Adaptive Interface (IAI) framework.

This chapter presents a formal definition of the problem and describes the solution. Information pertinent to the development of the intelligent agent and it's component parts are presented.

3.1 Problem Description

System operators can easily become overloaded by interfaces that require the operator to perform and monitor multiple distinct tasks. As the number of tasks an operator is given responsibility for increases, the amount of time they have available to spend on each decreases. Furthermore, the theory of cognitive load suggests that there is a limited amount of information we as humans can store in our working memory [22]. When the speed at which a system operates, or the number of tasks it requires the operator to perform increase, the workload the system places on the operator will increase as well. If the workload level rises to the point where it exceeds the operator's capabilities, the operator will become overloaded and the system's overall performance will begin to degrade.

At its core, the problem of mitigating overload in multi-tasking interface environments represents a classification problem. Given the state of the system and a set of inputs (features), action must be taken to mitigate the overload condition. The problem is therefore defined by its inputs and its action space.

3.2 Problem Definition

For a given system, operator, and interface, an IAI must take action such that the overall performance with regard to interface interactions is maximized.

Due to the stochastic nature of most operating environments, it is important to make the distinction between overall system performance and performance within the interface. Stochastic environments can impose a high level of variance into a system. Overall system performance can reflect this variance, and therefore be an unreliable indicator of the effectiveness of the IAI.

For IAI solutions to be truly adaptive, they must take into account the characteristic features of the operator. This problem can be compounded by the heterogeneous mix of operators for the given system. Operator skill levels may range from novice to expert. Actions that are beneficial for one skill level may be detrimental for another.

A solution to the problem is represented by a classification table for each operator that maps their performance, workload, and the current state of the interface to an action. This action should modify the state of the adaptive interface in a way that is expected to increase the combined performance of interface interactions.

3.3 Workload Measures

Before we can develop a mechanism to mitigate operator overload, we must first consider the different methods by which our system can determine operator workload levels. Traditionally, workload has been derived using one of three methods.

3.3.1 Subjective Workload. Subjective workload measures are obtained by asking the operator to rate what they believe the workload level was for a task on a given scale [25]. Subjective measures are highly regarded because they are thought to be fairly accurate and fool-proof. They do have downsides however. The person giving the subjective workload measure must explicitly participate in the measuring activity, which can take the form of a verbal or written questionnaire. The measure is dependent on the honesty of the respondent, whose motives may not be entirely clear. Additionally, the individual providing

the measure might be good at differentiating two different workload conditions, but they might not be able to accurately rate it on a scale against which future measures could be compared. Users often use mitigation stragegies (avoid, transfer, delay, simplify) to reduce workload, and this behavior will affect their ability to estimate workload. Subjective measures are also time consuming, and cannot usually be collected in real-time. This factor limits the quality and resolution the subjective measure can provide, especially in real-time system modification.

3.3.2 Performance Based Workload. Performance based workload is determined by analyzing an operator's performance at the activity they are performing [9]. Workload and performance usually have an inverse relationship. Poor performance can be indicative of high workload. Good performance can be indicative of low or manageable workload levels. The downside of using performance-based workload measures is that by the time workload increases have been detected, performance has already begun to suffer.

3.3.3 Physiometric Workload. A subject's physiological features can be used to determine their approximate workload level [48]. This measure is based on the fact that humans experience measurable physiological changes as workload levels increase. Most often, high workload levels induce stress and the fight-or-flight response. Workload increases can be expressed physiologically as increases in heart rate, galvanic skin response, and decreased pupil size. More complex measuring techniques can derive workload by tracking gaze patterns or by directly measuring brainwave activity with an electroencephalogram. Physiological measures may be able to give an indication of workload levels before the workload levels begin to have a significant impact on performance. Some physiological measuring devices can be invasive however, requiring the operator to wear electrodes and other measurement apparati. Even measures that are traditionally considered to be non-invasive, such as eye-tracking, can require that the operator sit or behave in a manner that allows the measurement device to consistently be able to track the operator's features. Additionally, some physiometric signals may be delayed or reactive rather than predictive.

3.4 Methods of Automation

The AIA will be responsible for setting the appropriate automation mode. Consideration must therefore be given to the automation scheme that the interface employs. Potential schemes for our agent testbed include automation by levels [12], automation by individual tasks [37], and mixed-mode automation.

3.4.1 Automation by Levels. One method by which tasks within the interface can be automated is by grouping them into Levels of Automation (LOA) [12]. The levels constitute a taxonomy that ranks the automation from full manual control to full automation. Each level specifies the different types of tasks that the automation will assume control over. These tasks can include planning, including analysis of alternatives, decision making, and implementation. The automation can also be mixed-initiative. In a mixed-initiative system, the agent proposes a course of action and waits for the operator's approval before implementing it. Under the LOA scheme, the system's automation level is simply a sliding scale. The agent's possible actions include increasing, decreasing, or leaving the automation level as is.

3.4.2 Automation by Individual Tasks. Another method of automating interface activities involves breaking the automation down by individual tasks [37]. Each task is then assigned a mode, either manual control or complete automation. This scheme allows more complex and custom permutations of automation activities, but comes at the expense of greater complexity in the agent's action space.

3.4.3 Mixed Mode Automation. Mixed mode automation is a hybrid between automation by levels and automation by individual tasks. With mixed mode automation, levels of automation can be applied to individual tasks or groups of similar or related tasks within the interface. This scheme has the advantage of offering higher fidelity in the assignment of automated activities, while at the same time reducing the complexity of the agent's action space.

3.5 An Agent-based Reinforcement Learning Approach

The relationship between an operator's workload and their combined system-operator performance is complex [24]. No single mapping between workload and performance can be applied to all system operators. Each operator brings certain traits, skill sets, and proficiency levels that make them unique. The mapping between workload and performance must therefore be learned for each operator and interface. In order to learn this mapping, the solution involves the use of a reinforcement learning agent. The reinforcement learning agent monitors the interactions of the operator and interface. It specifically attempts to learn when the introduction of automation improves performance for a given pre-existing workload level and performance level. Once the agent has learned the most appropriate times to introduce interface automation for a given operator, it will be capable of improving the overall system performance.

To test this hypothesis, we have developed the agent and an agent testbed (described in Chapter 4). The agent monitors task performance and overall workload for the testbed tasks. At certain time intervals, the agent is given the choice to increase, decrease, or leave automation levels untouched for each of the tasks in the testbed. By enabling automation within the interface, the agent is able to manipulate the workload and cognitive load levels for the operator. By increasing automation, when appropriate, the agent will be able to decrease the cognitive load of the operator. When automation is enabled for a particular task, the operator will be able to focus their attention on the remaining non-automated tasks. Their cognitive load will therefore be reduced. By allowing the operator to focus their effort on the tasks that they can efficiently manage, and automating those that they cannot, the agent-based approach will lead to increased overall system performance. The elements that define the artificial agent include performance measures, environment, actuators, and sensors (PEAS) as well as the method it uses to define a mapping between them [35].

3.5.1 Performance Measures. Overall system performance measures are generally task specific. Each type of system has different methods for measuring its own success. For stochastic environments, these measures are subject to variance imposed by the envi-

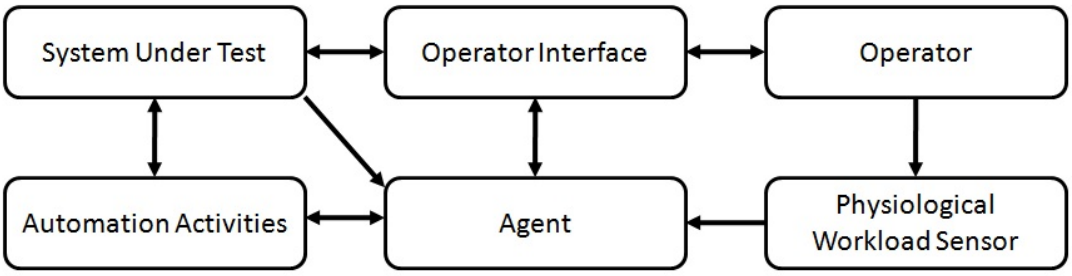


Figure 4 Agent-System Integration Diagram.

ronment. This variance poses a risk to our agent. If the variance is significant enough, it can influence the agent's behavior. To mitigate this variance, the performance measure for the interface agent must be derived from the system's overall performance without being directly linked to it. It is important that measures be normalized so that valid comparisons can be directly made between two separate values without regard to additional factors.

Our agent makes use of a simple performance ratio. The ratio provides a measure of the effectiveness that was achieved under the current adaptive scheme for a given time period compared to what could otherwise have been achieved under the best alternative scheme during that same time period.

3.5.2 Agent Environment. The agent exists in a hybrid environment. While it enables and disables automation activities within the operator interface, it does not entirely reside there. The inputs it receives come from system outputs and workload sensors that exist outside of the interface. Because the sensors that feed data to the agent may be imperfect, the overall environment is only partially observable.

The AIA acts between the operator interface and the system's array of automation activities, as shown in Figure 4. The operator interacts with the system through the operator interface. Data can also be collected from the operator through the use of physiometric sensors. This data, once properly correlated with the operator's workload level, can be used by the agent as a workload feature to select an action or automated mode. The agent, using this physiometric data along with workload data from the interface, chooses the appropriate mode to select for interface automation.

3.5.3 Agent Action Space. The interface agent affects the environment by implementing actions that are explicitly defined in its action space. The action space for the agent in a given state includes all possible automation modes available for transition from the current state. Our agent and agent testbed make use of the automation by tasks scheme.

3.5.4 Agent Sensory Inputs. The sensor inputs required by the agent to make automation decisions include the current state of system automation, the current system performance, and the operator's current workload level. Because our system uses multiple agents to reduce complexity, the workload and performance of the overall system must be accounted for in the state representation for each of the interface's individual agents.

3.5.5 Reinforcement Learning. Deciding when to enable or disable automation for a particular task is an important consideration for the agent. Because we are looking to augment the abilities of a human operator, rather than fully automate them, it must be true that the operator brings something to the table that full automation cannot provide. It may be that the automation is imperfect, and that the human operator is able to perform the task better than the automation. The automation may also be brittle. There may be certain failure modes to which it is particularly susceptible. Human operators are generally very robust, and unlike automation, can usually reason their way through unknown scenarios. Additionally, there may be tasks that the automation will not be allowed to perform because of safety, legal, or ethical concerns. For these reasons, the agent must strive to maintain the highest level of operator awareness and involvement.

Every operator is different. Each has different skill levels and different capabilities to handle varying workload levels. Additionally, the response each operator gives, both performance and physiological, varies from person to person. The agent must therefore be able to learn the different thresholds for each operator and provide an automation response that is most appropriate for the operator.

Given the preceding considerations, the agent uses a reinforcement learning approach to learn the best action to take for a given operator at a given time. The Q-Learning

$$State(n) \leftarrow AState(n) * Perf(n) * otherAState * otherPerf * Workload$$

$$AState(n) = \begin{cases} 1 & \text{Manual} \\ 2 & \text{Automated} \end{cases}$$

$$Perf(n) = \{1, 2, 3\}$$

$$otherAState = round\left[\left(\sum AState - AState(n)\right)/(n-1)\right]$$

$$otherPerf = round\left[\left(\sum Perf - Perf(n)\right)/(n-1)\right]$$

$$Workload = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

Figure 5 Agent state function.

procedural control algorithm is depicted in Algorithm 1. The agent's state is defined by the current state of the automation activities, the operator's current physiological workload indications, and the operator's current performance, as depicted in Table 3 and defined in Figure 5.

Figure 5 shows the five values that define each agent's state. $AState(n)$ represents the agent's state, manual or automated. $Perf(n)$ is a performance value, discretized from 1 to 3, that represents how well the tasks that the agent is responsible for are being performed. *otherAState* is a rounded average of the automation states for all other agents in the system. *otherPerf* is a rounded average of the performance values for all other agents in the system. The final state component, *Workload* is a value representing the overall system workload discretized from 1 to 10.

Table 3 Agent state representation specification.

Panel 1		Panel 2		Panel 3		Panel 4		Workload
AS	Perf	AS	Perf	AS	Perf	AS	Perf	
1..2	1..3	1..2	1..3	1..2	1..3	1..2	1..3	1..10

For each state that is encountered, the agent is given the option of taking actions that may or may not increase overall system performance. The action, coupled with the state from which it was made, is known as a state-action pair. The total number of state-action

pairs is equal to the sum of the number of actions for every possible state. The action space for the agent consists of two discrete actions, enable or disable automation.

As actions are taken in the environment, a reinforcement learner records the performance level that is achieved for each state-action pair. This recorded knowledge can then be exploited at a later point. If the agent reaches a state for which possible state-action pairs have been previously explored, the agent can exploit that information to choose the best action that historically yielded the best performance.

If a system is highly dynamic, the agent must occasionally forgo the exploitation of historic results to detect change and determine the new results that each state-action pair will achieve. This is done by using a probabilistic model to determine when the agent should try a random action (exploration) and when it should take action based on what historically was the best course of action (exploitation).

Algorithm 1 Q-Learning Procedural Control Algorithm [43].

INITIALIZE $Q(s, a)$ arbitrarily
Repeat (for each episode):
    Initialize s
    Repeat (for each step of episode):
        Choose a from s using policy derived from Q
        Take action a, observe r, s'
        $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$
        $s \leftarrow s'$
    until s is terminal

3.5.6 Implications of Exploration. For a reinforcement agent to learn, it must explore the outcomes of its state-action pairs. Exploration occurs when the agent chooses to take an action other than what it has determined was previously the best possible action for a given state. This can take place randomly, according to a probabilistic model, or when the agent reaches a state for which no actions have previously been explored.

When the agent is first initialized, it does not possess any knowledge about which actions are appropriate for a given automation state and sensor input. Therefore, all initial actions in the system are exploratory, and essentially random. As the agent works to explore state-action sequences, it gains knowledge that it can exploit when the state

is encountered again. This period for the agent is known as the initial learning phase. For a system involving a human operator this phase may be especially frustrating. The agent will explore all courses of action, including those with poor outcomes, until the best possible state-action pairs have been identified and can be exploited.

Because the agent plays a role in a dynamic system, it must be continuously learning so that it can adapt to change. These changes can be the result of new system performance or mission profiles. The operator can also change. As their proficiency with the system improves, the automation assistance they receive must adapt. An automation style that had been appropriate at a lower skill level may no longer be appropriate once they become more experienced. For a reinforcement learner, continuation learning takes place randomly. The frequency of this continuation learning is predefined in the system. These random actions may not always produce positive results.

3.5.7 State Change Intervals. When the interface automation state is changed there will be a period during which the workload and performance levels change and then stabilize. It is important that the agent allow enough time for the values to stabilize before recording the outcome of the previous state change. If the values are not allowed to stabilize, outcomes for state-action pairs will not be accurate. Only after the values have stabilized, and the results of the prior state change have been recorded, can further state change opportunities be evaluated. Based on the literature evaluated in Section 2.4, the state change interval selected for our agent is 30 seconds.

3.5.8 Keeping the Human in the Loop. If the overall performance of the interface automation exceeds the performance achieved when the operator directly manipulates the interface, the agent may learn that the best course of action is to fully automate the system. This behavior would violate one of our core goals, which is the need to keep the human operator in the loop as much as possible. It may therefore be necessary to pad the performance measure of the human operator such that the human operator is rewarded with a higher performance value than the automation.

3.5.9 Agent Process. The agent process, based on the Q-Learner RL paradigm, is listed in Algorithm 2. The agent repeatedly selects the best available action for the given state, implements the selected action, and then waits 30 seconds to determine the outcome. Once the outcome is determined the agent determines the next state, based on the new system state. The outcome from the previous state choice is also saved in the Q table using the reward function from Figure 6. This process repeats continuously until the system's exit conditions are met.

Algorithm 2 Agent Process Pseudocode.

```
INITIALIZE QLearner
while Not ExitCondition do
  CurrentAction ⇐ QLearner.getAction(CurrentState)
  IMPLEMENT CurrentAction
  DELAY 30 Seconds
  nextState ⇐ getState(AutomationState, Performance, OtherAutomationStates,
                OtherPerformance, Workload)
  Reward ⇐ Performance
  QLearner.update(CurrentState, NextState, CurrentAction, Reward)
  CurrentState ← NextState
end while
```

3.5.10 Agent Reward Function. The agent's reward function, described in Figure 6, determines what value will be passed on and accumulated in the Q table for the prior state-action pair. The reward function for the agent is designed with a penalty coefficient, δ, for automation activities. This coefficient serves to bias the reward function such that manual operation is preferred to automation by a margin of δ. The actual performance for each activity is reported to the agent as a decimal percentage between 0 and 1. The value represents the ratio of the current performance for the task out of the maximum possible performance. Scoring calculations for the activities in the testbed are established in Chapter 4.

3.5.11 Accelerated Learning Strategies. The initial learning phase of a reinforcement learner is characterized by instability. The random actions taken by the agent while it learns will negatively affect the performance of the overall system. Depending on the complexity of the state space and numbers of possible actions, this negative affect could

$$Reward \leftarrow (actualPerformance - 1) - ((AState(n) - 1) * \delta$$

$$actualPerformance = \{0..1\}$$

$$AState(n) = \begin{cases} 1 & \text{Manual} \\ 2 & \text{Automated} \end{cases}$$

$$\delta = AutomationPenaltyCoefficient$$

Figure 6 Agent reward function.

last for a significant period of time. If absolutely nothing were known about the agent environment, then this period of poor performance would be a necessary cost.

Fortunately, we are not completely oblivious to the nature of the agent environment. By analyzing the characteristics of an operator's performance and workload features during a calibration, certain assumptions can be made that may help to approximate when automation state transitions would be appropriate. These assumptions can be used to pre-populate the knowledge of the reinforcement learner. The pre-populated values are unlikely to be perfect, however they may help to mitigate the performance decrease that is usually present while the agent is in its infancy. Eventually, the pre-populated values will become insignificant as the agent continues to learn from the actual environment.

One of the main benefits of the reinforcement learning approach to this problem is that it creates an agent capable of offering personalized automation assistance to the operators. Because of the different skill levels and preferences of the operators, it is expected that there will be variability between agents for different operators. It is likely however that using another operator's agent will produce better results during the early training stages than would be achieved by training a new agent from scratch. This presents another possible source for data to pre-populate the learning agent.

4. Experimental Methodology

To test the Automation Mode Selection (AMS) agent's performance, a simulated multi-tasking interface environment is required. This environment, named Workload and Automation Level Response Simulator (WALRuS), will be used to gather experimental data from human operators performing simple tasks with variable workload levels. Because the testbed will not include a feedback mechanism to allow the operator to report their actual workload, the workload value, referenced henceforth, is the user's perceived workload as derived from their task load. The simulation will record the operator's performance for each perceived workload level to which the operator is subjected. The simulation will allow for several modes, testing the subject's performance both with and without the use of the automation mode selection agent. For a single subject, comparisons can then be made showing potential performance differences with the addition of the AMS agent. Data collected will also be used to build a performance profile for each subject. This data will be used as a source of offline training material for the agent.

4.1 System Engineering Methodology

4.1.1 Requirements and Architecture. The overarching system is composed of two main components. The first component, described in Chapter 3, is the AMS agent. The second component is the AMS testbed, WALRuS. WALRuS will be used to build a performance profile for human subjects. This will be accomplished by testing subjects with workload inducing tasks and measuring their performance. The performance profile will provide a measure of the performance a subject is capable of achieving for a given workload, over a range of workload levels. The AMS agent shall be capable of using this performance profile as training data to determine LOA thresholds, thereby mapping workload levels to specific Levels of Automation (LOAs). This mapping will then be imported by the AMS testbed, allowing the simulated performance achieved through the use of the AMS's mapping to be determined through the use of automated trials.

4.1.2 System Verification and Validation. Interface trials conducted in WALRuS will include descriptive event and data logging for all simulation events. By logging all

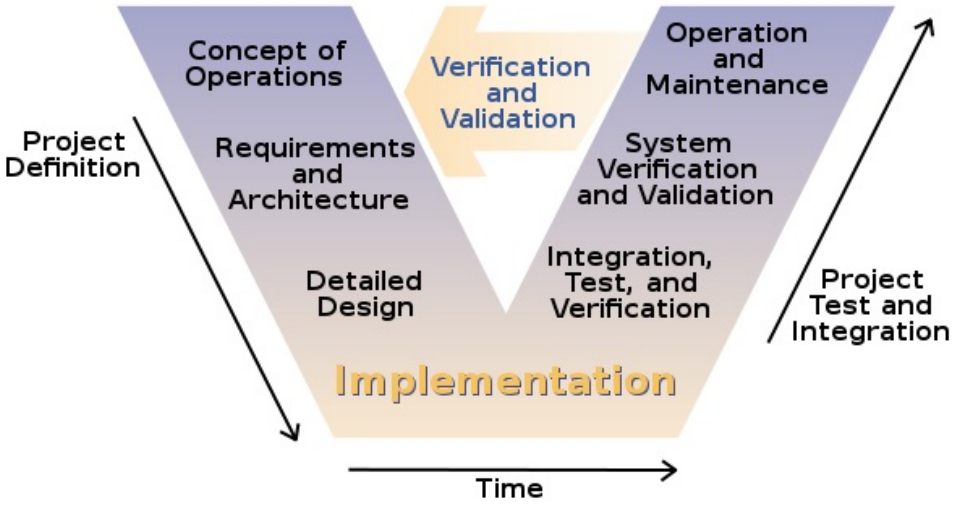


Figure 7 Systems Engineering Process, V Model [30].

events, verification and validation can be achieved by analyzing data logs post-trial and matching actual LOA changes with those specified by the AMS agent's output.

4.1.3 Operation and Maintenance. The AMS testbed will be operated in two modes. The first mode is the profile building mode. This mode will not incorporate any automated functionality, nor will it be used to test any portion of the AMS agent. The purpose of the profile building mode is to construct a performance versus workload profile for a single specific human subject. The second mode is the AMS agent performance test mode. This mode will incorporate two LOAs. The workload and performance thresholds for these LOAs will be determined by the output of the AMS agent. The purpose of this mode is to build a second performance versus workload profile that will allow comparative performance to be analyzed.

4.2 Requirements Specification

The testbed environment's interface shall be designed to conform to certain pre-defined requirements to effectively meet the needs of the research. Each of these requirements are broken down, defined, and analyzed in the requirements specification sections below.

4.2.1 Hardware Requirements. The testbed simulation environment shall run on a hardware platform running the Windows 7 operating system. The system shall accept inputs from the operator through the use of keyboard and mouse input devices. The simulation shall operate in full-screen mode, with only the minimum required visual information relevant to the simulation being displayed to the subject.

4.2.2 Interface Complexity. The subject demographic for this study includes Air Force enlisted, officer, and civil service personnel. No specific technological or operation background will be required. It is expected that subjects will be familiar with the use of standard "point-and-click" interface operations. The interface will therefore not be targeted toward any operational system or platform, and instead represent generic tasks that a subject can be trained to perform in a short amount of time. Additionally, the simulation and interface must be simple enough that it can be properly implemented and tested in order to meet the time line available for the study.

4.2.3 Multi-Tasking Environment. The impetus for the development of the AMS agent comes from operator performance and overload issues that arise in the MAC and UAV communities. These issues are a direct result of demands placed on the operator by multiple simulateous tasks. Therefore, the environment for which the interface is developed shall include elements that test the operator's ability to multi-task, or manage different tasks, each with their own separate interface elements, simultaneously. The individual tasks shall also be representative of the types of tasks performed by UAV operators. These tasks may include, but are not limited to, system monitoring, tracking, communications, and resource management activities.

4.2.4 Data Collection Rate. The agent described in Chapter 3, Agent Development, requires up-to-date workload and performance data to determine automation states and to update the agent's table of expected utility. To meet this need, the simulation shall provide workload and performance data to the agent at a rate equal to, or better than, the agent's state change interval. The minimum value for the state change interval is expected

to be 15 seconds. Therefore, to ensure appropriately filtered data is available to meet this requirement, performance data shall be collected at a rate of 1 Hz or better.

4.2.5 Workload Variability. To properly test the operator's response to varying workload levels, the simulation shall be capable of varying the operator's task loading. This task loading, which will be referred to as the simulation workload level, shall be quantifiable on a discrete scale from 1 to 10. The workload level should correlate to actual operator task loading in a linear fashion. While the workload level shall be recorded and used as input to the AMS agent, it shall not be made visible to subjects of the experiment.

4.2.6 Physiometric Workload Measurement. While the simulation workload level can provide a value indicating the operator's task loading, it does not provide any insight into the operator's cognitive loading. Therefore, the testbed shall include provisions for the measurement and recording of at least one physiometric workload indicator. Some potential candidates include EEG, Galvanic Skin Response (GSR), Heart Rate (HR), and Eye Blink Rate (EBR). The physiometric indicator is expected to correlate to their cognitive loading. This value should provide insight into the operator's mental state at the varying workload levels.

Provisions for the selected physiometric indicator should be accounted for when designing other aspects of the simulation. Some physiometric sensors may require the operator to sit in a certain spot without moving. Others may inhibit the use of hands and arms, requiring consideration when designing the methods by which the operator is expected to input data for the experiment.

4.3 Multi-Attribute Task Battery

In 1992, psychology researchers Arnegard and Comstock devised an experiment called the Multiple Attribute Task Battery (MATB) [2]. The MATB was developed as an experimental testbed to analyze the performance of operators while they performed multiple simultaneous tasks. The set of tasks was designed to be representative of the types of

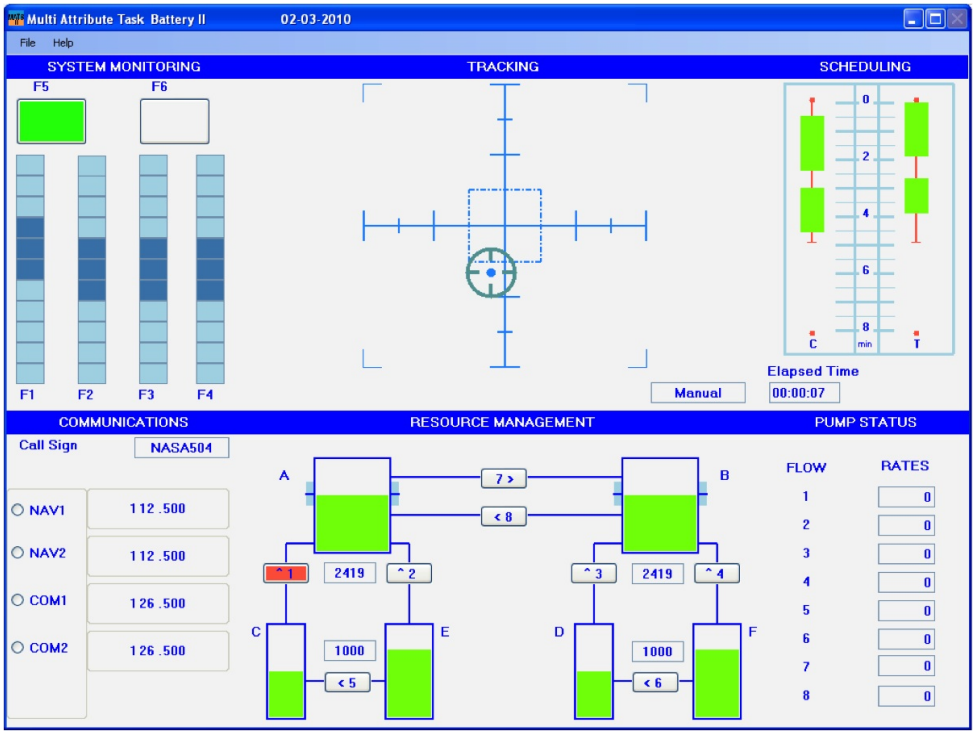


Figure 8 MATB-II experimental interface testbed developed by NASA [26].

tasks that are performed by pilots, air crew, and other equipment operators. The testbed allowed experimental interface elements to be tested in a controlled environment.

Researchers at the National Aeronautics and Space Administration (NASA) modernized the experiment in 2003 by creating the MATB-I and later the MATB-II [26]. The experiment includes tasks that test the operator's ability to perform system monitoring, tracking, communications, and resource management tasks. As shown in Figure 8, the MATB layout includes a main screen with subpanels for each of the tasks the experiment's subject is expected to perform. NASA's MATB-II includes provisions for both visual and auditory alerts and instructions.

The MATB meets most of the requirements for our interface testbed. It is also an accepted and well-known tool in the psychology community for testing experimental user interface designs. These factors make the MATB style of experiment testbed an ideal candidate to form the basis for our experimental testbed. Unfortunately, NASA's MATB-II platform does not meet all of the requirements for our testbed. In order to meet these

requirements, the testbed will be custom built and given a visual interface that closely resembles the MATB style of experiment.

4.4 Developing an Agent Testbed

To test the effectiveness of the agent-based reinforcement learning approach on interface automation mode selection, a battery of tasks representative of Multiple Aircraft Control (MAC) has been selected. This task battery will exercise the operator's ability to manage high workload scenarios by presenting the operator with a range of workload levels. The rate at which the operator is subjected to tasking is used to increase their workload level. The rate will be adjusted beyond the point at which they would normally become task saturated in an unaided state. Unlike the MATB, this testbed uses a single mode of data representation and response. The sections below describe the development of our simulation testbed, named Workload and Automation Level Response Simulator (WALRuS), and enumerates the design features that allow it to meet the system's requirements.

4.4.1 Target Platform. In order to streamline the testbed development process, the Python programming language was selected as the primary language for the testbed implementation [33]. Python offers many advantages which make it ideal for this type of project. Python's main advantage is that it is portable. Code written in Python is stored as text-based scripts, rather than compiled binaries. While the target system is the Windows 7 operating system, the Python code developed for this project can be executed on other platforms, such as Mac OS and Linux, without modification. Python is also free to use and is licensed as open source software. This ensures that other parties will be abe to effectively duplicate this research, or re-use the testbed for future research.

In addition to Python's other appealing attributes, one of its greatest benefits is the wide availability of Python libraries. These libraries extend the language and give programmers access to extra hardware and software capabilities. In order to display graphical interface elements, WALRuS makes use of the Pygame graphics library [32]. Like Python, Pygame is licensed as open source software and is developed and maintained by members of an online community. Pygame offers advantages that help speed up graphics processing,

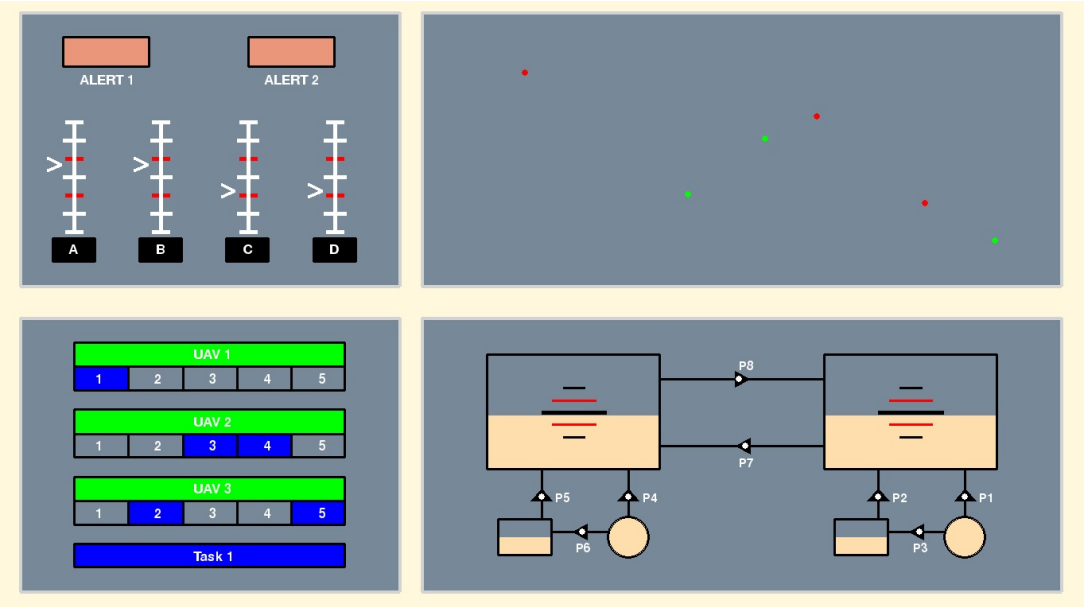


Figure 9 WALRuS Full Operator Interface Screen.

such as hardware graphics acceleration and multi-core CPU support. It also incorporates functions to make drawing graphical elements easier from a programmer's perspective, and includes functionality necessary for the acquisition of operator input from the mouse and keyboard.

4.4.2 Interface Tasks. The tasks WALRuS presents to the operator are broken down into four independent subpanels. None of the tasks are co-dependent, and inputs and operator performance in any one panel have no effect on the events that occur in the other three. The four tasks are modeled after the types of tasks included in NASA's implementation of the MATB. Each is intented to be representative of activities performed by pilots and flight crew. Minor modifications have been made to increase the rate at which events occur, for which high-resolution operator performance can be measured, as well as changes to make the tasks more representative of the activities performed by operators of unmanned aircraft. The four tasks selected for the WALRuS testbed are depicted in Figure 9. These include system monitoring, object tracking, vehicle assignment, and resource management tasks.

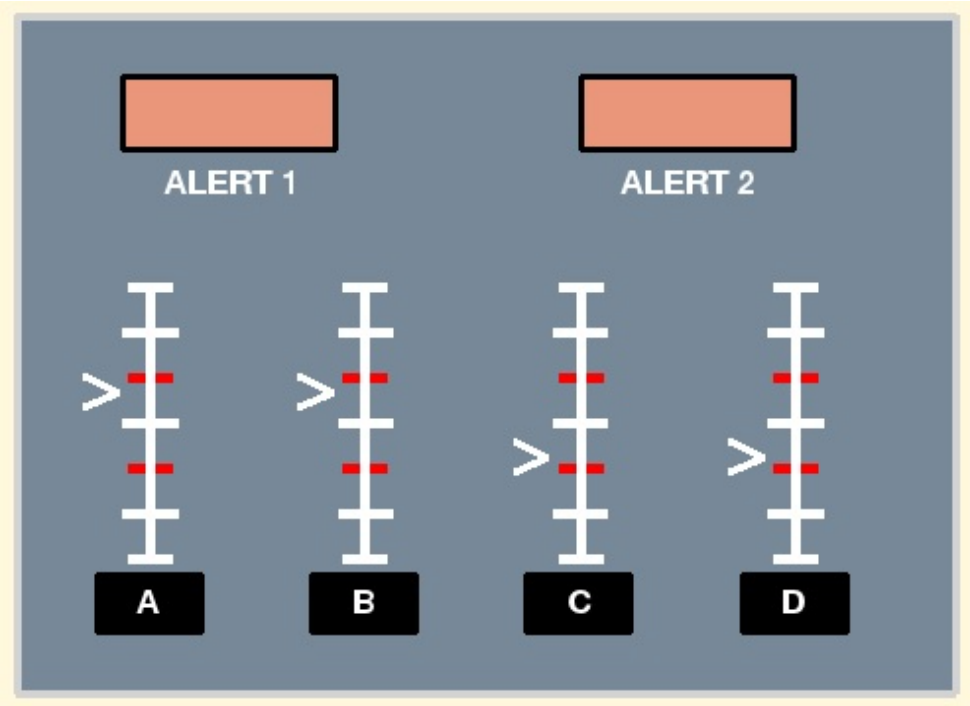


Figure 10 WALRuS System Monitoring Task Panel.

4.4.2.1 System Monitoring Task. The system monitoring task, shown in Figure 10, occupies the upper left quadrant of the testbed's interface. This task consists of six separate subtasks. Of the six, four are depicted as level indicators and two are depicted as simple alert boxes.

The four level indicators trend up and down as the simulation progresses. The speed at which the indicators move is a function of the simulation's workload level. When the indicator reaches or exceeds the portion of the bar marked with red graduations, the subject must left-click any portion of the indicator bar or label to designate it as an anomaly. Clicking on the bar causes its label to turn red, as shown in Figure 11. If the bar's indicator is not outside of its normal range, clicking on it will not result in an anomaly designation. When the indicator on a designated bar moves back into the normal range the anomly designation is automatically removed.

The two alert boxes at the top of the system monitoring panel are triggered throughout the simulation. When the alert is triggered the box turns red, as shown in Figure 12.

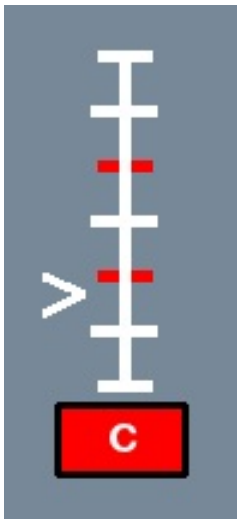


Figure 11 WALRuS System Monitoring Task Panel: Designating Indicator Errors.

Figure 12 WALRuS System Monitoring Task Panel: Triggered Alert Box.

When this occurs, the subject must left-click the box to acknowledge the alert. When the alert is acknowledged the background color returns to the un-triggered state.

Scoring for the system monitoring task panel is recorded at a rate of 60Hz. Calculations for the task's total score are shown in Figure 13. Each of the six subtasks are weighted equally. Each of the level indicators is awarded a score of 1 if they are within their normal operating range, or if they are outside of their normal operating range and designated. If the indicator is outside of the normal operating range without being designated, the subtask is awarded a score of 0. The alert boxes are awarded a score of 1 when they are not illuminated, and a score of 0 when they are illuminated bright red. The combined score for the panel is the sum of these individual scores divided by the number of subtasks.

$$Indicator(n) = \begin{cases} 0 & \text{Alert Range, Not Designated} \\ 1 & \text{Alert Range, Designated} \\ 1 & \text{Normal Range} \end{cases}$$

$$AlertBox(n) = \begin{cases} 0 & \text{Illuminated} \\ 1 & \text{Not Illuminated} \end{cases}$$

$$Score = \frac{\sum Indicator + \sum AlertBox}{numElements}$$

Figure 13 Monitoring Task Scoring.

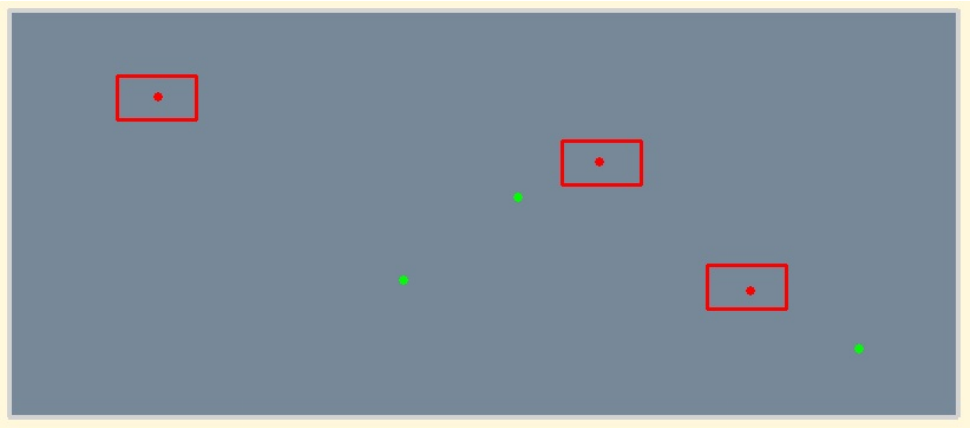


Figure 14 WALRuS Tracking Task Panel.

4.4.2.2 Tracking Task. The tracking task, shown in Figure 14, occupies the upper right quadrant of the testbed's interface. This panel shows six colored target circles, three of which are red, and three of which are green. The three red target circles constantly move in a linear fashion. The speed that the target circles move is a function of the workload level. Higher workload levels cause the target circles to move faster, while lower workload levels cause target circles to move slower.

The subject is responsible for keeping the red targets designated. Designating a target is accomplished by clicking on or near it, causing a designator box to appear as shown in Figure 15. The designator boxes do not move. When the target leaves the designator box it is removed. The subject can place an unlimited number of designator boxes around a single target so long as the target intersects the designator box.

Scoring for the tracking task panel is recorded at a rate of 60Hz. Calculations for the task's total score are shown in Figure 16. Each of the targets is assigned a score of

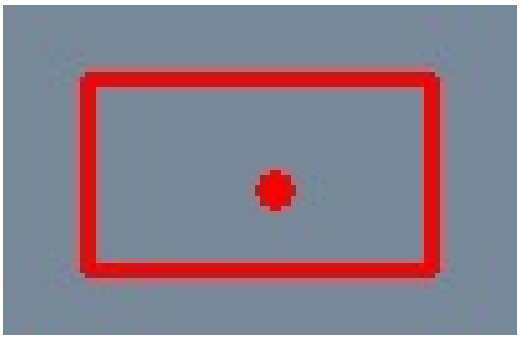


Figure 15 WALRuS Tracking Task Panel: Target Designator.

$$Target(n) = \begin{cases} 0 & \text{Not Designated} \\ 1 & \text{Designated} \end{cases}$$

$$Score = \frac{\sum Target}{numTargets}$$

Figure 16 Tracking Task Scoring.

1 if it is properly designated. If it is not designated, the target is assigned a score of 0. The combined score for this panel is the sum of the target scores divided by the number of targets.

4.4.2.3 Vehicle Assignment Task. The vehicle assignment task, shown in Figure 17, occupies the lower left quadrant of the testbed's interface. The panel depicts the mission packages for three UAVs and a vehicle tasking.

For the vehicle assignment task, the subject is responsible for designating a vehicle that has the appropriate mission package to accomplish the the displayed tasking. For example, if the the current tasking is "Task 3", the subject must select a UAV that has "3" highlighted in its mission package display. The union of all UAV's mission packages is equal to the tasking superset. Therefore, it is possible that more than one UAV will have the appropriate mission package, but it will never be the case that none of the UAVs have the appropriate mission package. If more than one UAV is capable of performing the required tasking, the subject may choose either without penalty.

To select a UAV, the subject must simply click anywhere in the UAV's listing box. Selecting the UAV will cause its box to illuminate, as shown in Figure 18. The subject

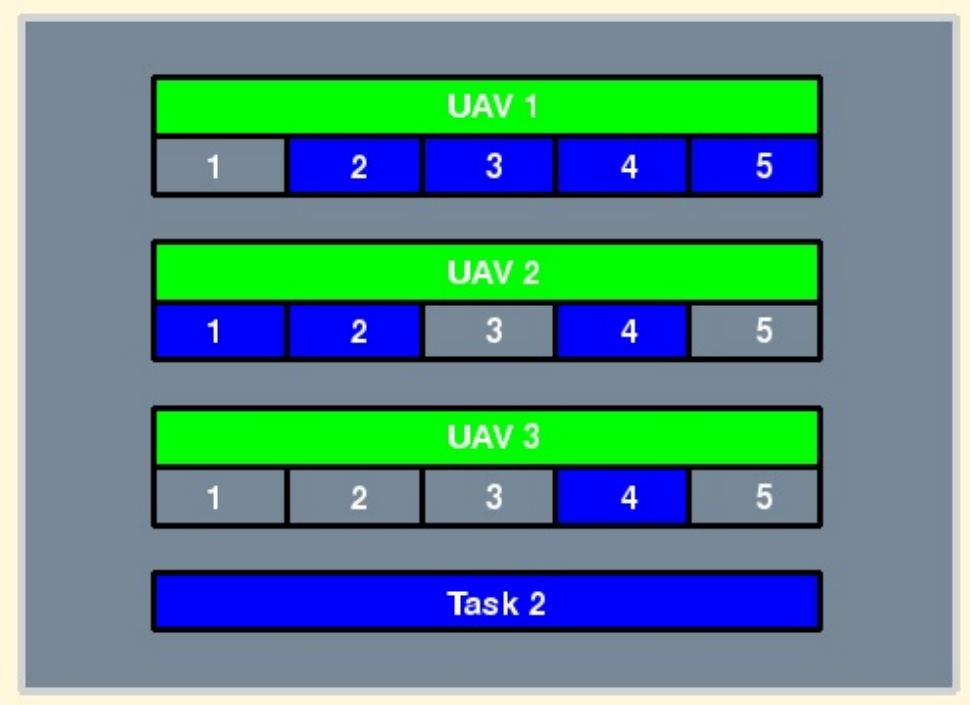


Figure 17 WALRuS Vehicle Assignemnt Task Panel.

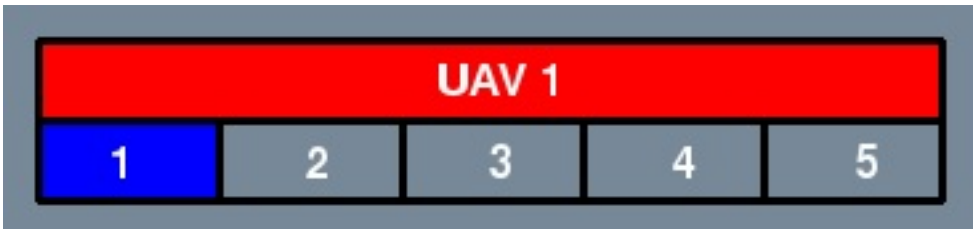


Figure 18 WALRuS Vehicle Assignemnt Task Panel: UAV Designation.

$$Score = \begin{cases} 0 & \text{No Task Assignment} \\ 0 & \text{Invalid Task Assignment} \\ 1 & \text{Valid Task Assignment} \end{cases}$$

Figure 19 Vehicle Assignment Task Scoring.

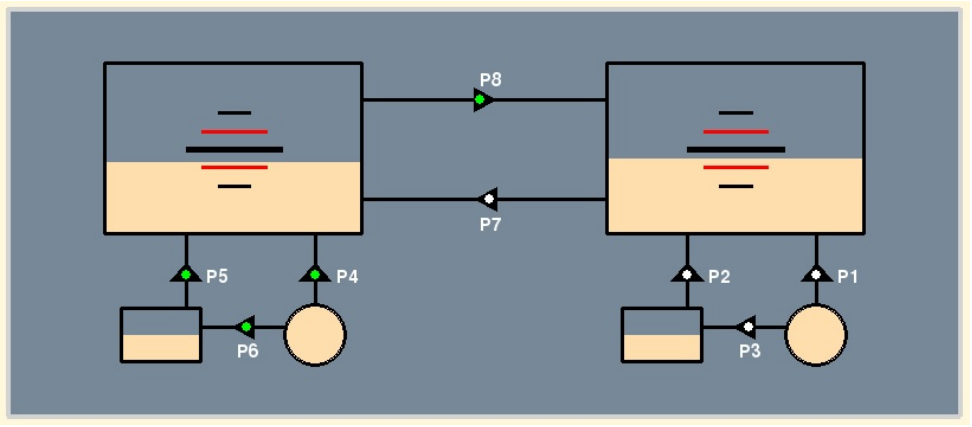
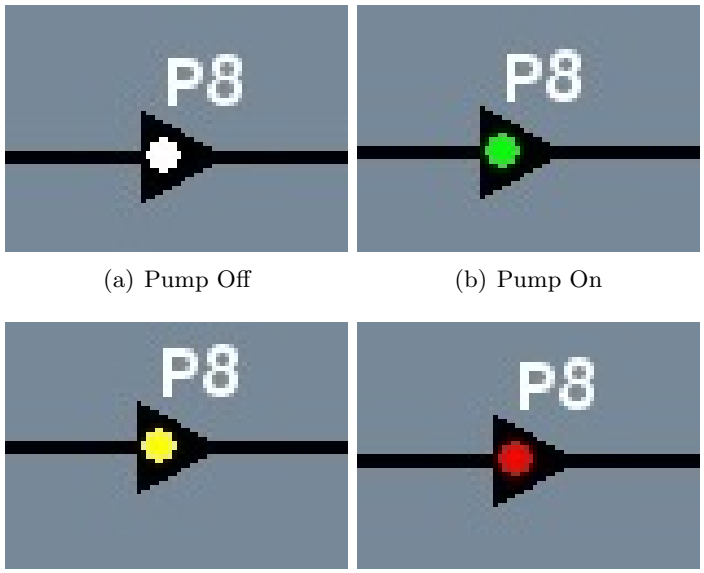


Figure 20 WALRuS Resource Management Task Panel.

can change the designated UAV at any time. When a new tasking arrives, the mission packages and UAV designations are reset. The speed at which taskings arrive is a function of the simulation's workload level. Lower workload levels cause taskings to arrive at a slower rate, while higher workload levels cause taskings to arrive at a faster rate.

Scoring for the vehicle assignment task panel is recorded at a rate of 60Hz. Calculations for the task's total score are shown in Figure 19. If an appropriate UAV is designated for the current tasking, the overall task is awarded a score of 1. If no UAV is designated, or if a UAV is incorrectly designated, the overall task is awarded a score of 0.

4.4.2.4 Resource Management Task. The resource management task, shown in Figure 10, occupies the lower right quadrant of the testbed's interface. This tasks panel consists of two identical subtasks arranged horizontally. The task is represented as a set of tanks, pumps, and reservoirs. The objective for the subject is to maintain the levels in the two largest tanks such that they do not exceed the upper or lower bounds marked by red graduations.



(a) Pump Off (b) Pump On

(c) Overheat Warning (d) Overheat Disabled

Figure 21 WALRuS Resource Management Task: Pump Status Indicators.

To achieve the task's objective, the subject must enable or disable certain combinations of the panel's eight pumps. The pumps consist of black triangles with small inset circles representing the pump's current status. Pumps can be enabled or disabled by clicking on or relatively near the pump. When enabled, the pump will transfer from one reservoir or tank to another. The direction of flow is the direction in which the triangle is pointing. With the exception of the circular reservoirs, the pump will lower the level in the source tank and raise the level in the destination tank. The circular reservoirs can never run dry, and therefore pumps connected to them have an unlimited charge source.

To ensure that the task does not become a "set and forget" type of task, the pumps are set to overheat and fail if they are left on for too long. After a short period of time, that varies with workload level, enabled pumps will begin to overheat. When this happens the pump's status indicator will change to yellow. The operator is given several seconds to disable the pump before it overheats. When the pump overheats, the status indicator changes to red. After this occurs the subject must wait for the pump to cool before its status indicator changes back to white, and it can be restarted. The various status indicators are depicted in Figure 21.

$$Tank(n) = \begin{cases} 0 & \text{Level Exceeds Graduations} \\ 1 & \text{Level Within Graduations} \end{cases}$$

$$Score = \frac{\sum Tank}{numTanks}$$

Figure 22 Resource Management Task Scoring.

Scoring for the resource management task panel is recorded at a rate of 60Hz. Calculations for the task's total score are shown in Figure 22. Each of the two main tank subtasks are weighted equally. For each reporting cycle, the tank scores are reported as 1 or 0. If the tank's level indicator falls between the red graduations, the score is recorded as 1. If the tank's level indicator is above or below the red graduations, the score is recorded as 0.

4.4.2.5 Scenario Generation. The scenario generated by the testbed for each experiment is created according to a pseudorandom event schedule. Each of the task panels is designed such that event triggers are unaffected in any way by the operator's actions. The occurrence of events in the system monitoring task, tracking task, and vehicle assignment task occur according to the pseudorandom schedule. The operator is simply along for the ride, much like a game of 'Whack-A-Mole'. The resource management task panel is slightly different. The flow rates of each component in the system are set by the testbed, however the operator is completely in control of pump operations and pump response is on a fixed timetable that relates directly to the system's workload level.

The advantage of the pseudorandom event schedule is that it allows each of the experiment's human subjects to experience the exact same scenario. It also allows more flexibility, in that events do not need to be explicity scheduled by researchers. Explicitly assigned schedules are time consuming to create, especially if multiple scenarios are required. Under the pseudorandom scenario generation scheme, different schedules can be created simply by changing the Random Number Generator's (RNG) seed value, which is assigned in the parameter section at beginning of the testbed's Python code.

4.4.3 Workload Variability. The operator's workload in WALRuS is manipulated by increasing or decreasing the speed at which events occur in each of the four task panels.

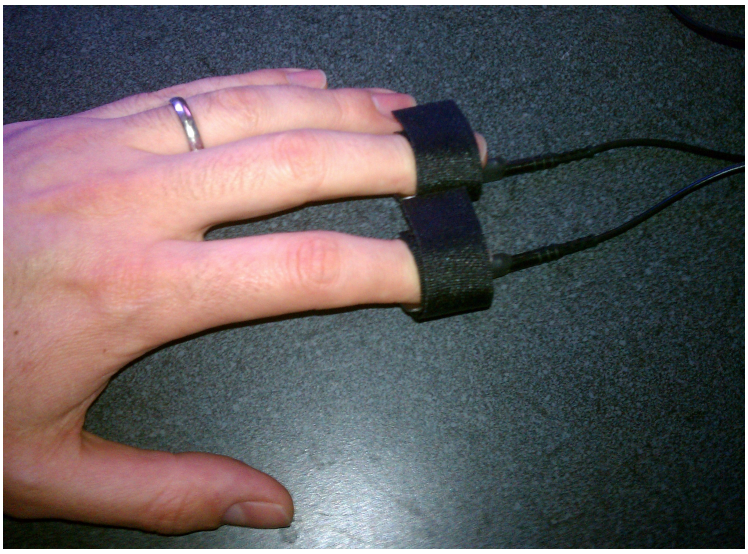


Figure 23 Galvanic Skin Response (GSR) Finger Electrodes.

The testbed is configured for ten discrete workload levels. The simulation uses a single workload level to control the speed of all four task panels. The levels represent a linear relationship to event speed.

4.4.4 Physiometric Workload Measurement. To obtain a biological measure of the operator's workload the system measures and records physiometric data from the subject. The physiological indicator used by WALRuS is Galvanic Skin Response (GSR). Galvanic Skin Response (GSR) is a measure of skin conductivity. High workload scenarios evoke stress, causing the subject's eccrine glands to excrete perspiration. Prior studies have shown that GSR correlates well with workload [40].

In order to measure the subject's GSR, WALRuS requries that the subject wear a monitoring device while participating in the experiment. The stainless steel finger electrodes, shown in Figure 23, are worn on the middle and index fingers of the subject's non-dominant hand. As the subject begins to perspire, the conductivity between the electrodes and the subject's fingers increases. The circuit, shown in Figure 24, is used to measure this change. The breadboard shown in Figure 25 is the physical implementation of this circuit. Changes in conductivity measured across the electrodes is amplified by the circuit and measured by an Analog to Digital Converter (ADC). The value is then passed

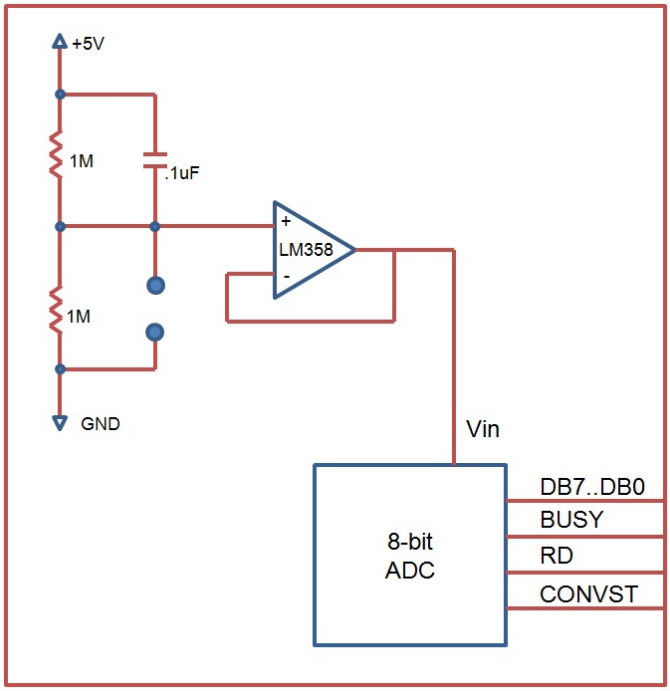


Figure 24 Galvanic Skin Response (GSR) Circuit Schematic.

as input to a microcontroller (uC) where it is serialized and transmitted to the computer running WALRuS.

While recording, the hand to which the GSR electrodes are connected must remain relatively motionless. Large movements can cause the electrodes to shift, creating noise in the recording data. Because of this, the GSR electrodes effectively immobilize the subjects non-dominant hand. The dominant hand is therefore left as the subject's sole means of providing input to WALRuS. This factor was taken into consideration while implementing each of the four task panels. While other implementations of the MATB make use of both keyboard and mouse inputs [25], WALRuS obtains all input through mouse movements and mouse clicks.

4.4.5 Data Collection. The WALRuS testbed operates at 60Hz, storing one data sample per cycle. Data is stored in a comma delimited file, with one value recorded per cycle for each of the interface's subtasks. Storing data at these rates allows for greater

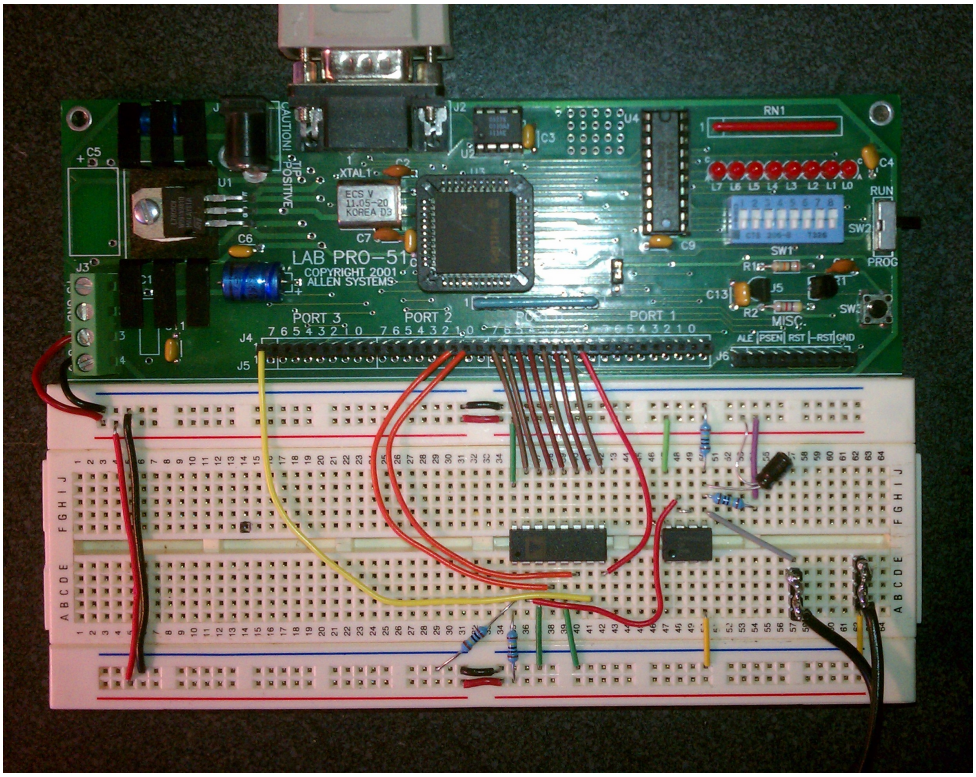


Figure 25 Galvanic Skin Response (GSR) Meter Board.

flexibility when processing the data after the experiment is complete. Data files written by WALRuS are then read in by the AMS agent and used for offline training.

4.5 Evaluating Agent Performance

To verify our hypothesis, it must be shown that the agent-based learning approach results in improved overall system performance. At a minimum, the agent-based approach must not produce worse results than would be possible if it were not present in the interface. Additionally, the agent must produce gains that are significant enough to outweigh the additional costs and constraints its implementation imposes on the system.

4.5.1 Action Space Convergence. One of the weaknesses faced by reinforcement learners is the time that must be spent to train the agent. For an interface agent, this learning phase usually requires the active involvement of the human operator. Each state transition represents a learning opportunity for the agent. In our system, state transitions are time constrained by the minimum interval required for performance and workload measures to stabilize. The number of state-action pairs that can be explored is equal to the number of possible states multiplied by the number of actions that can be performed from each state. High resolution state spaces with high action counts therefore lead to massively long learning times for the agent. These long learning times can make human trials very tedious.

4.5.2 Agent-Based Performance Evaluation. Using human subjects is undeniably the most accurate way to test our interface agent, but the numbers of subjects and time required to achieve statistically significant results may make using human subjects impractical. As an alternative, cognitively modeled agents can assume the role of the human operator. Computer-based agents have the advantage of not becoming fatigued or distracted. They also do not complain when they are subjected to experiments for long periods of time. Removing humans from long-duration trials also allows for the stabilizing time interval to be eliminated, thereby increasing the number of trials that can be conducted in a given amount of time.

Figure 26 WALRuS Transition Screen.

4.5.3 Human Study. To test the performance and convergence times for AMS, agent training must be accomplished with long-duration human trials or with accelerated trials using the cognitively modeled agents described in section 4.5.2. This study uses the accelerated method. Source data for the cognitive agents must still be generated. To do this, WALRuS is used to build a profile correlating a subject's workload levels to the performance they are able to achieve for each of the four task panels. To build a performance profile for an individual subject, WALRuS iterates through the ten workload levels. The subject's performance is recorded and averaged over the course of a five minute trial at each workload setting. At the end of each trial, the screen shown in Figure 26 is displayed. The subject is allowed a short break to prevent eye and muscle strain. During this time the screen, shown in Figure 26, is presented. By right-clicking the computer's mouse, the subject is able to resume the experiment and begin the next, higher workload, scenario.

4.5.4 Test for Maximum Sustainable Workload. One tool devised for the WALRuS testbed to determining the effectiveness of our approach versus a full manual system is the Maximum Sustainable Workload Test (MSWT). The MSWT is shown in Algorithm 3. This test is inspired by TCP Reno [23], a network congestion avoidance algorithm for Transmission Control Protocol (TCP) packets. TCP Reno adjusts network transmission rates to maintain the highest possible transmission rate without packet loss. In much the

same way, the MSWT adjusts workload levels in order to achieve and maintain the highest possible workload for a given performance.

Algorithm 3 Maximum Sustainable Workload Test Pseudocode.

```
MPCount ← 0
Mode ← FS
while Not ExitCondition do
  if Performance < ρ then
    MPCount ← MPCount + 1
    if MPCount >= α then
      Threshold ← Workload * ϵ
      Workload ← Threshold
      Mode ← OA
    end if
  else
    MPCount ← 0
    if Mode = OA then
      Workload = Workload + δ
    end if
    if Mode = FS then
      Workload = Workload * γ
    end if
  end if
  DELAY β Seconds
end while
```

4.6 Summary

This chapter documents the development and implementation of the Workload and Automation Level Response Simulator (WALRuS) testbed. Design considerations and operator instructions for each of the interface's four panels is presented. Additionally, methods used to generate scenarios and record data from experimental trials is discussed. Finally, a method of using the WALRuS testbed to determine maximum sustainable workload is presented.

5. Analysis and Results

The Workload and Automation Level Response Simulator (WALRuS) testbed was subjected to a limited human study including five subjects. Study participants were scheduled for ninety minute blocks, of which a total of sixty minutes were used to conduct actual interface trials. The purpose of these trials was to gather performance data for the subject at different workload levels. Participation included a ten minute training period followed by ten separate five minute trials. Prior to beginning the trials, subjects were informed about the purpose of the study, the nature of the data collected, and of their right to opt out of the study at any time. Subject training was conducted by the research author and consisted of one five minute session explaining the operation of the individual tasks, followed by one five minute practice session to allow the subject to gain experience and become familiar with the tasks. Once the practice sessions were complete, the first actual trial was loaded.

The subjects were instructed to proceed at their leisure. By right-clicking the introduction screen, the subject would begin the first timed five minute trial. Once each trial completed, the load screen would automatically reappear and tell the subject what the number of the next trial would be. Subjects were allowed necessary time between trials to rest their eyes and relax. The research author monitored all subjects throughout the testing phase.

Each of the ten individual trials were conducted at a different workload setting. The first trial began at the lowest possible workload setting and proceeded to work up to the highest possible setting by the last trial. Each subject was assigned a unique subject identification number, and data from all ten of their trials was stored in a single file as described in Section 4.4.5.

5.1 Subject Performance Profiles

Once a subject had completed the series of ramped workload inducing exercises, their raw data files were processed to create profiles that matched the subject's workload level with a performance level. This workload and performance correlation is later used

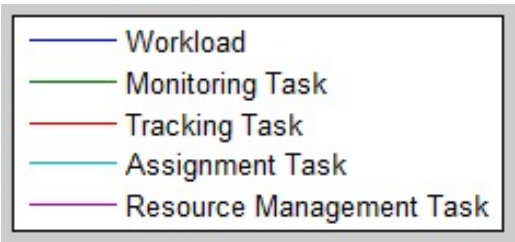


Figure 27 Performance Profile Legend.

to determine how well a subject could be expected to perform given a particular task workload. The unfiltered data, depicted in Figure 28, shows the performance of each of the five subjects over the course of the experiment. The legend for the data is included in Figure 27. The increasing workload, depicted by the blue stepped line, shows a relatively clear inverse relationship to subject performance for the four task panels.

The performance data for each of the ten trials was then processed to provide an average performance for each of the workload levels. This data is graphed in Figure 29 for each of the five subjects. The legend for this data is depicted in Figure 27. Figure 29 shows the inverse relationship between workload and performance.

5.2 Agent Response Convergence

One of this study's main goals was to determine if a reinforcement learner's action space would mature towards convergence within a reasonable amount of time. For the domain, twelve hours of training was selected as a value to be considered a reasonable learning period. This period represents the amount of time an operator could be expected to use the system over the course of two working days. In order to test the agent's convergence times, the agent was repeatedly assigned data from the the output of the profile building scenarios. The action selected by the agent was then plotted, as shown in Figure 30. The results shown in the figure are from a single subject, but are typical of the study population.

The results in Figure 30 are arranged in a four by ten grid. Each column represents the action selected by the four agents for a given workload. As one moves to the right in the listed columns, the workload level increases. Each subgraph shows the agent response as it

converges to a value. The x-axis represents the number of learning iterations, while the y-axis shows the agent's response (manual(1) or automated(2)). The results from numerous runs indicates that each agent typically converges within 160 iterations. In Section 2.4, we determined that thirty seconds would be selected as our system's cycle time. This indicates that the expected convergence time during an actual human trial could be as low as eighty minutes for each of the potential workload levels. For our system, with 10 distinct workload levels, the time for complete convergence would be 13.3 hours. This value assumes that each of the possible workload levels are represented equally throughout the agent's training period. In practical use, it is unlikely for this to be the case.

5.3 Cognitive Agent Trial Results

The performance profiles from Section 5.1 formed the basis for the development of our cognitive agent. These profiles, acting as a Look Up Table (LUT), allowed the Automation Mode Selection (AMS) agent to report the expected performance of the operator for a given workload level. Using the agent response table created in Section 5.2, the behavior of the AMS agent could be determined. These elements allowed a simulated comparative performance evaluation to be conducted. Each evaluation consisted of a simulated one hour (120 cycle) test run using the cognitive agent. The results of a typical evaluation for the data collected from Subject 2 are displayed in Figure 32, with the legend being displayed in Figure 31.

The workload level that is used to generate each scenario, shown in blue, is created by a one dimensional pseudo-random walk algorithm. The un-aided performance reported by the cognitive agent is shown in black, while the expected performance of the overall system with automation is shown in red. The response of each of the four agents is shown in the lower half of Figure 32. As the workload increases, a decrease in the cognitive agent's performance can be seen. Workload changes that trigger automation mode switches for each of the four agents can also be seen. Increases in workload generally lead to the agents switching from manual to automated modes, as would be expected. Decreases in workload generally lead to agents switching from automated to manual modes, also as would be expected. The aided system performance exceeds the un-aided performance in all trials.

As expected, the two performance values are equal when the agents disable all automation activities.

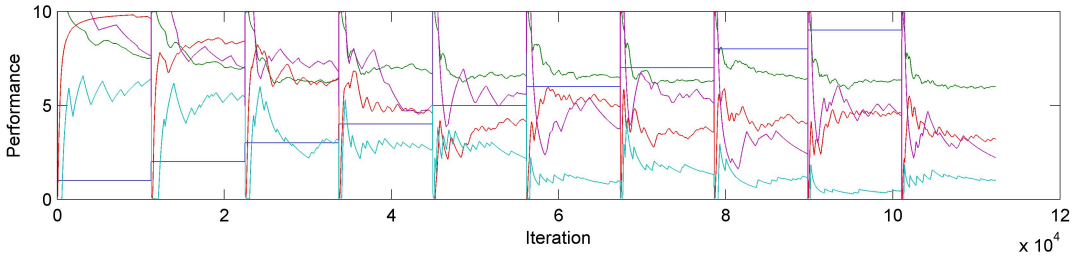
5.4 Physiometric Workload

The Galvanic Skin Response (GSR) sensor, described in Section 4.4.4, proved to be troublesome during the initial testing trials. The Analog to Digital Converter (ADC) circuit that provided data to the microcontroller (uC) was unable to reliably relay data. The exact cause is still unknown, but it is believed to be a timing issue with the Analog Devices ADC chip. A replacement ADC box was obtained and connected to the system, however the additional power requirements it imposed on the circuit damaged the development board's voltage regulator. Due to timing constraints, and the lack of immediately available replacement hardware, the GSR recordings were eliminated from the testing regimen at the last minute.

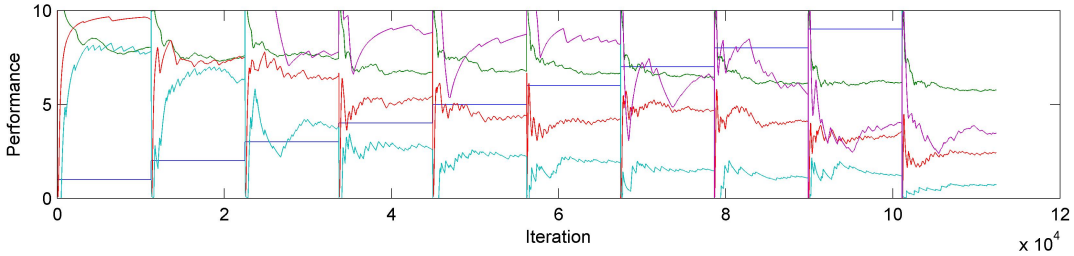
Ultimately, the practicality of using a device such as the GSR meter as a source for physiometric workload data is called into question. Hardware issues aside, the device does require subjects to immobilize one hand. With only the use of a single hand remaining, the operator is left with a diminished capacity for physical interaction with the system. It is unlikely that such a setup would be acceptable on an actual vehicle control console.

5.5 Summary

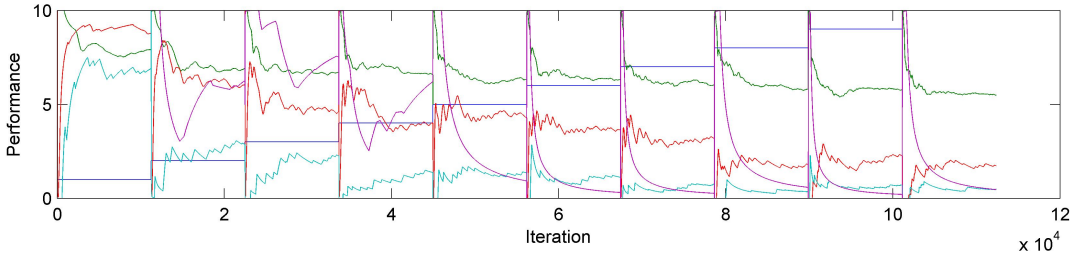
In order to test the performance and convergence time of the Automation Mode Selection (AMS) agent, cognitive agents were built using data collected from a series of limited human trials. These trials tested the level of performance each subject was able to achieve in the Workload and Automation Level Response Simulator (WALRuS) testbed for a range of workload levels. The results were used to create a performance profile LUT that served as the primary data source for each cognitive agent. The cognitive agents were then exposed to scenarios created by a pseudo-random workload generator. The performance of the cognitive agent, both with and without the assistance of the AMS agent, were then displayed for comparison.



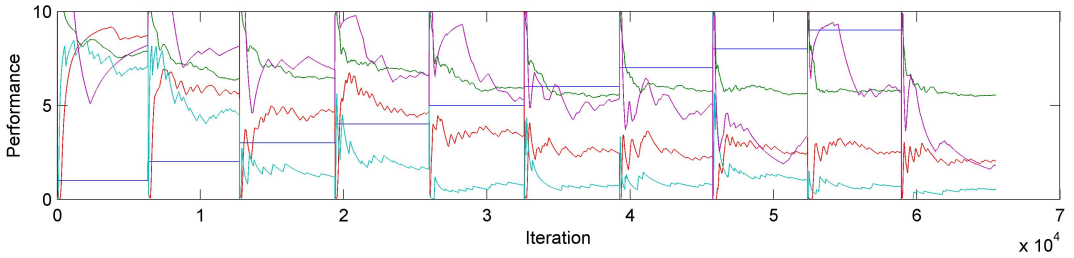
(a) Subject 1



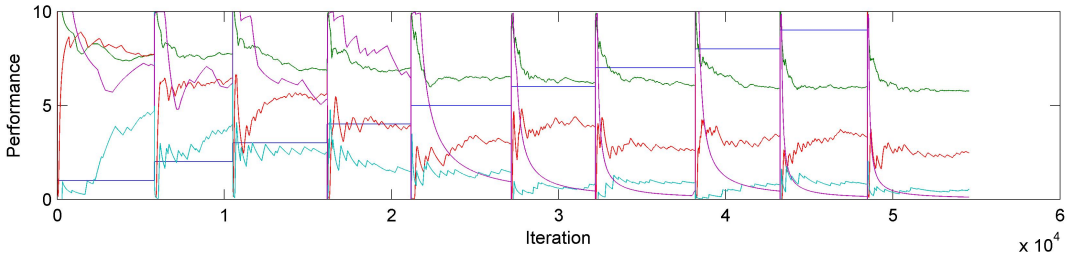
(b) Subject 2



(c) Subject 3

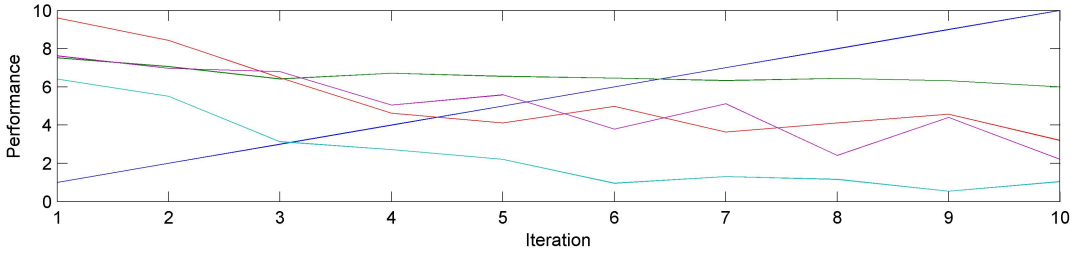


(d) Subject 4

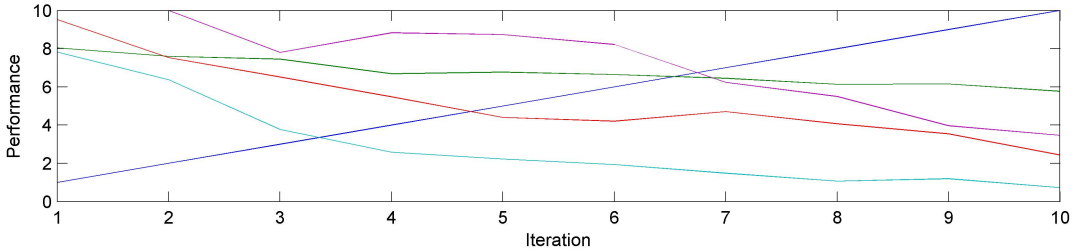


(e) Subject 5

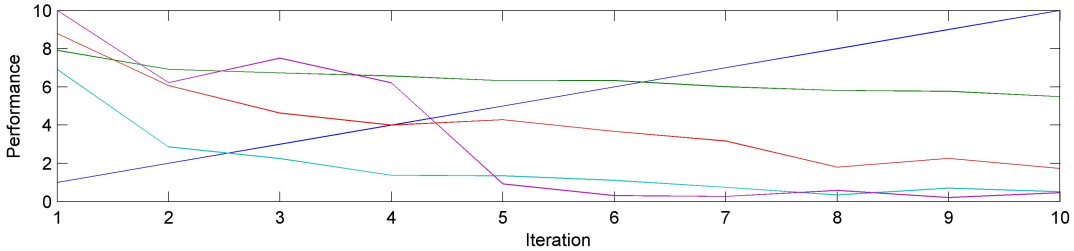
Figure 28 Subject Performance Profiles (Raw Data).



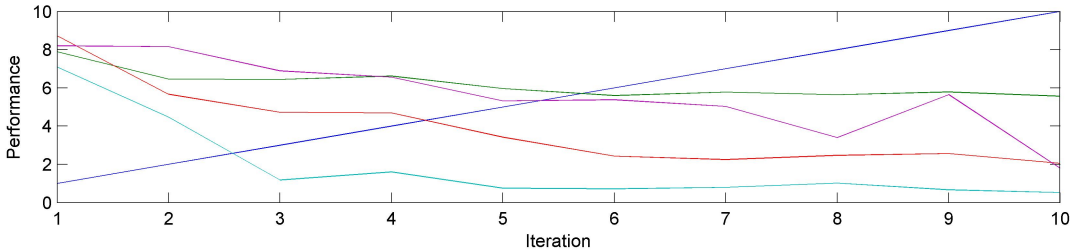
(a) Subject 1



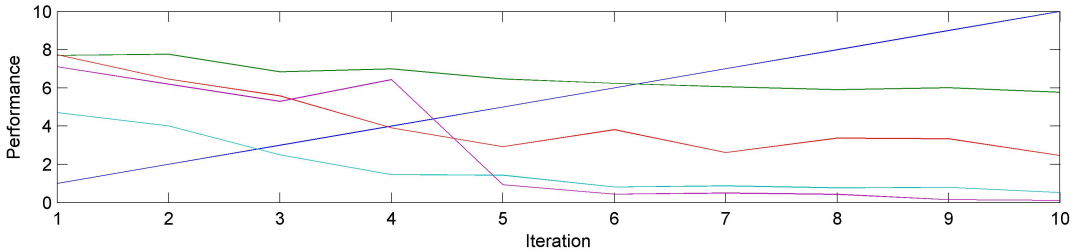
(b) Subject 2



(c) Subject 3



(d) Subject 4



(e) Subject 5

Figure 29 Subject Performance Profiles (Averages).

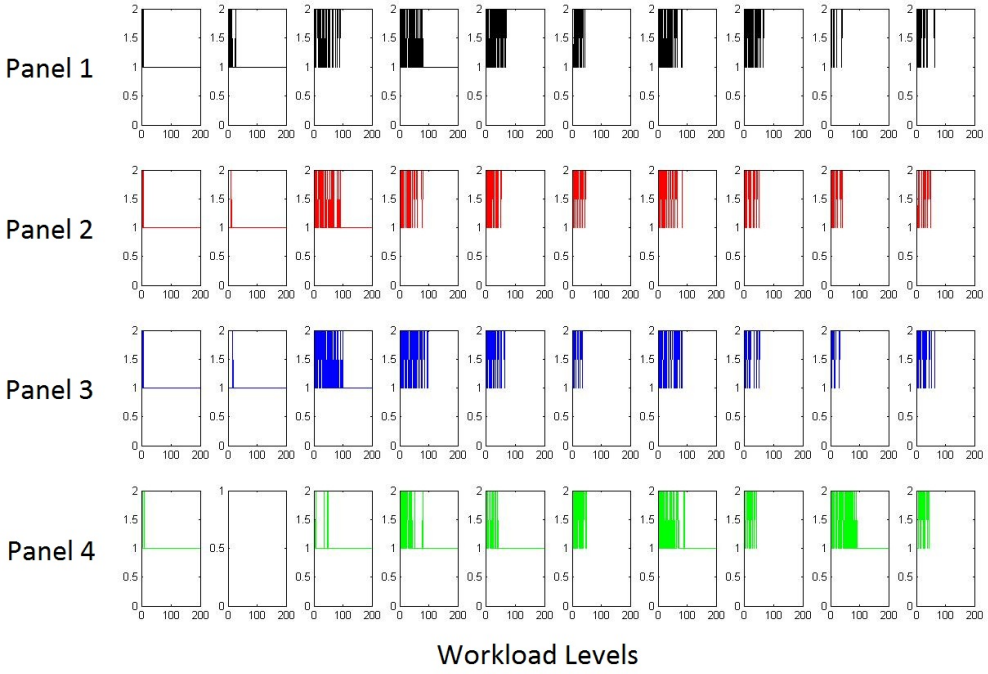


Figure 30 Typical AMS Agent Convergence Cycles.

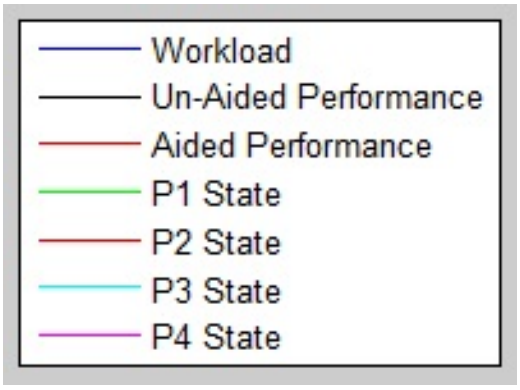


Figure 31 Cognitive Agent Performance Legend.

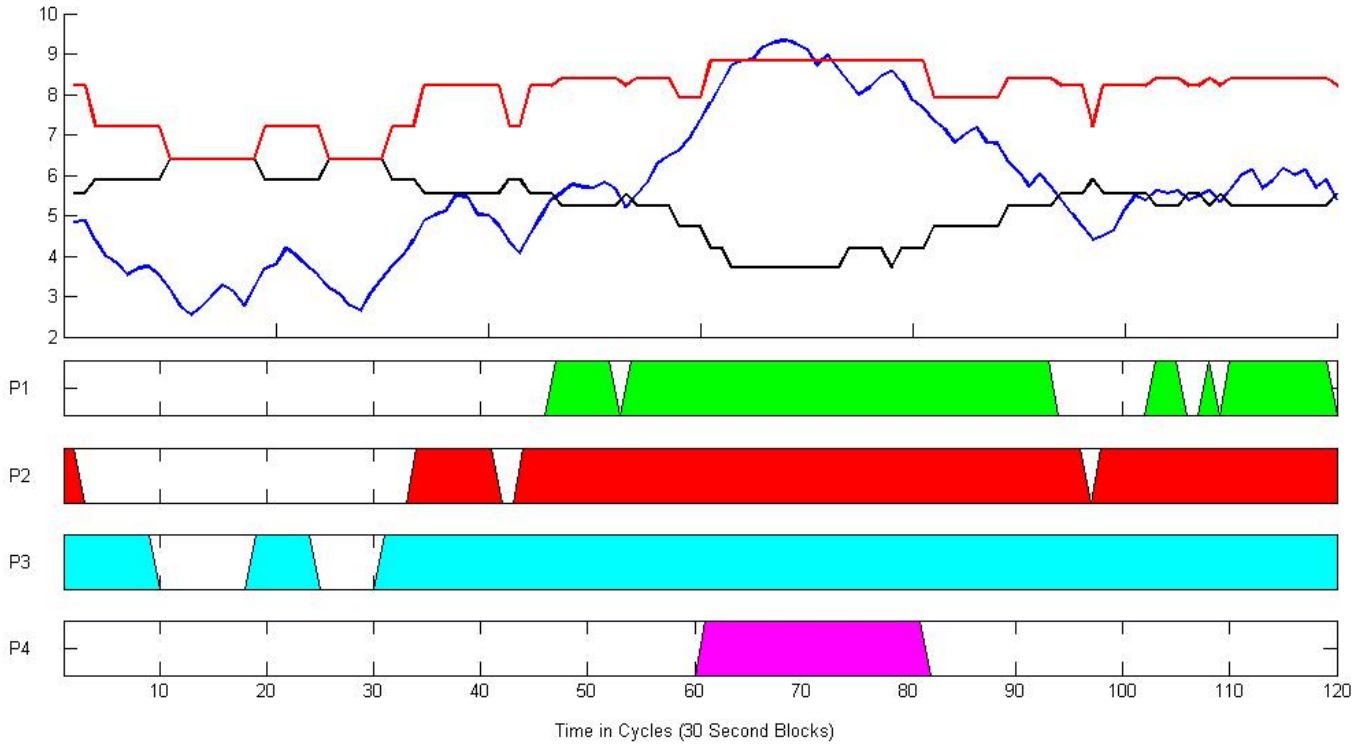


Figure 32 Typical Performance Comparison for Cognitive Agent.

6. Conclusions and Recommendations

The experiments conducted throughout the course of this research have demonstrated the feasibility of the intelligent AMS agent concept. Using the four agent model, the experimental convergence times achieved with the agents came within a small margin of meeting the goals set as reasonable and efficient. By using either simplified agent state representations, or a more advanced agent learning paradigm, it is likely that the convergence times would have surpassed these goals.

Performance profiles created through the use of the study's human trials have also confirmed that workload and performance do not possess a simple relationship. Results from the study data show a complex non-linear relationship that is different for each subject. This finding confirms the hypothesis that a 'one size fits all' approach to automation mode selection would inevitably produce inferior results.

6.1 Significance of Research

As the USAF continues to pursue MAC for its UAV fleet, steps must be taken to ensure that aircraft will still be able to be safely controlled under all potential conditions. Operator overload is one of the most significant hurdles that the MAC concept has yet to overcome. Automation has been tagged a a potential remedy for operator overload, but choosing the best way to implement automation while still maintaining the operator's situation awareness and preventing operator dependence are largely unanswered questions.

This research proposes new ways to mitigate the effects of operator overload and improve situation awareness by introducing an intelligent agent into the system interface. The intelligent agent learns by monitoring an operator's interactions with the system, and chooses when the most appropriate times are to implement interface automation. This research serves as a proof of concept and initial feasibility study to demonstrate the concept and determine potential agent training times.

Each operator possesses different skill and experience levels. Using a 'one size fits all' automation schedule to assist operators is certain to be sub-optimal. The intelligent AMS agent concept has the potential to improve operator efficiency by introducing automation to

an interface at the most appropriate time for a given operator. The agent-based approach also adds the advantage that it can change along with the proficiency of the operator.

6.2 Recommendations for Future Research

While using the system's workload and operator's performance levels as features to train the agent has shown promise, there may be further gains to be made through the use of physiometric data as additional training features. Future research should focus on the use of non-invasive or minimally-invasive physiometric sensors to include additional biological features to provide insight about the cognitive state of the operator.

When using performance as the sole means of detecting operator overload, the system must first detect a decrease in performance before it can try to correct it. This means that performance degradation is unavoidable. Adding perceived workload gives the system a means of predicting the operator's cognitive state, and is not susceptible to the same lag time that performance features are. By using physiometric data, the agent is given even more insight into the actual cognitive state of the operator. Having access to the operator's cognitive state means the agent will be able to detect and mitigate impending performance degradations that are caused by distractions, fatigue, and other conditions that exist outside of the system itself.

6.3 Summary

The purpose of this research was to investigate, develop, and test new methods for mitigating the negative effects of high workload scenarios, while increasing the performance and operator situation awareness for systems in the MAC domain. A literature review was conducted to determine the current state of research in this field, and to identify potential gaps. The identified gaps highlighted the need for better methods of determining how and when to introduce interface automation.

To fill this requirement, the intelligent AMS agent was proposed. This agent-based approach, designed to natively adapt to different operators and task applications, is laid

out in Chapter 3. WALRuS, an agent testbed, was developed to test the AMS agent using requirements and processes specified in Chapter 4.

A limited human trial was conducted to determine the feasibility and required agent training times. This study, documented in Chapter 4, provided data and results that show promise for the intelligent AMS agent concept.

Appendix A. Institutional Review Board Waiver Letter

DEPARTMENT OF THE AIR FORCE
AIR FORCE INSTITUTE OF TECHNOLOGY
WRIGHT-PATTERSON AIR FORCE BASE OHIO

29 November 2011

MEMORANDUM FOR MAJ KENNARD LAVIERS

FROM: William A. Cunningham, Ph.D.
AFIT IRB Research Reviewer
2950 Hobson Way
Wright-Patterson AFB, OH 45433-7765

SUBJECT: Approval for exemption request from human experimentation requirements (32 CFR 219, DoDD 3216.2 and AFI 40-402) for Command and Control Interface Automation Study.

1. Your request was based on the Code of Federal Regulations, title 32, part 219, section 101, paragraph (b) (2) Research activities that involve the use of educational tests (cognitive, diagnostic, aptitude, achievement), survey procedures, interview procedures, or observation of public behavior unless: (i) Information obtained is recorded in such a manner that human subjects can be identified, directly or through identifiers linked to the subjects; and (ii) Any disclosure of the human subjects' responses outside the research could reasonably place the subjects at risk of criminal or civil liability or be damaging to the subjects' financial standing, employability, or reputation.

2. Your study qualifies for this exemption because you are not collecting sensitive data, which could reasonably damage the subjects' financial standing, employability, or reputation. Further, the demographic data you are collecting cannot realistically be expected to map a given response to a specific subject.

3. This determination pertains only to the Federal, Department of Defense, and Air Force regulations that govern the use of human subjects in research. Further, if a subject's future response reasonably places them at risk of criminal or civil liability or is damaging to their financial standing, employability, or reputation, you are required to file an adverse event report with this office immediately.

WILLIAM A. CUNNINGHAM, PH.D.
AFIT Research Reviewer

```
# Copyright (c) <2012> <Air Force Institute of Technology>

# Permission is hereby granted, free of charge, to any person
# obtaining a copy of this software and associated documentation
# files (the "Software"), to deal in the Software without
# restriction, including without limitation the rights to use,
# copy, modify, merge, publish, distribute, sublicense, and/or
# sell copies of the Software, and to permit persons to whom
# the Software is furnished to do so, subject to the following
# conditions:

# The above copyright notice and this permission notice shall be
# included in all copies or substantial portions of the Software.

# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
# EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
# OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
# NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
# WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
# FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
# OTHER DEALINGS IN THE SOFTWARE.


import pygame,sys,os,time,random,math,pygame.gfxdraw,serial
from pygame.locals import *

pygame.init()
```

```
fpsClock = pygame.time.Clock()

#screenWidth = 1366
#screenHeight = 768

screenWidth = 1280
screenHeight = 800

simState = 0
workloadLevel = 0
cycleCountDown = 0

# ==================
# SIMULATOR SETTINGS
# ==================

trialTime = 180000
simSeed = 2

# ==================
# ==================
# ==================

screen = pygame.display.set_mode((screenWidth, screenHeight),
    pygame.FULLSCREEN | pygame.HWSURFACE | pygame.DOUBLEBUF)
#screen = pygame.display.set_mode((screenWidth, screenHeight),
    pygame.HWSURFACE | pygame.DOUBLEBUF)
pygame.display.set_caption('pyWorkload')

random.seed(simSeed)
```

```
redColor = pygame.Color(255,0,0)
greenColor = pygame.Color(0,255,0)
blueColor = pygame.Color(0,0,255)
whiteColor = pygame.Color(255,255,255)
blackColor = pygame.Color(0,0,0)
chargeColor = pygame.Color(255,222,173)
pinkColor = pygame.Color(255,105,180)
yellowColor = pygame.Color(255,255,0)

borderColor = (211,211,211)
frameColor = (119,136,153)

def pw(inVal):
  return int(screenWidth*inVal/100)

def ph(inVal):
  return int(screenHeight*inVal/100)

def arect(x,y,w,h):
  return pygame.Rect(pw(x),ph(y),pw(w),ph(h))

def nrect(old_rect):
  return pygame.Rect(pw(old_rect[0]),ph(old_rect[1]),pw(old_rect
      [2]),ph(old_rect[3]))

def apr((x,y)):
  return (pw(x),ph(y))
```

```
# ++++++++++++++++++++++++++++++++++++++++++++++++
# ------------------------------------------------
# Panel 1: Level Indicators
# ------------------------------------------------
# ++++++++++++++++++++++++++++++++++++++++++++++++

p1rect = arect(2,2,35,45)

gaugeValue = [0,0,0,0]
gaugeTarget = [0,0,0,0]
gaugeSelect = [0,0,0,0]

gaugeError = [0,0,0,0]

alertTime = [random.randint(10000,30000),random.randint
    (10000,30000)]
alertVal = [0,0]

def updateAlerts(in_tock):
  global alertTime,alertVal,workloadLevel

  tock = in_tock * workloadLevel * 0.75

  for i in range(0,2):
    alertTime[i] = alertTime[i] - tock
    if alertTime[i] <= 0:
      alertVal[i] = 1
      alertTime[i] = random.randint(10000,30000)
  return 0
```

```
def updateGauges(in_tock):
  global gaugeValue,gaugeTarget,gaugeSelect,workloadLevel,
     gaugeError

  tock = in_tock * workloadLevel * 0.15

  for i in range(0,4):
    if gaugeValue[i] == gaugeTarget[i]:
      gaugeTarget[i] = random.gauss(0,1)
      if gaugeTarget[i] < -3:
        gaugeTarget[i] = -3
      elif gaugeTarget[i] > 3:
        gaugeTarget[i] = 3
    elif gaugeValue[i] < gaugeTarget[i]:
      moVal = tock*0.0005
      if moVal > math.fabs(gaugeTarget[i]-gaugeValue[i]):
        gaugeValue[i] = gaugeValue[i] + math.fabs(gaugeTarget[i]-
           gaugeValue[i])
      else:
        gaugeValue[i] = gaugeValue[i] + moVal
    elif gaugeValue[i] > gaugeTarget[i]:
      moVal = tock*0.0005
      if moVal > math.fabs(gaugeTarget[i]-gaugeValue[i]):
        gaugeValue[i] = gaugeValue[i] - math.fabs(gaugeTarget[i]-
           gaugeValue[i])
      else:
        gaugeValue[i] = gaugeValue[i] - moVal
    if gaugeValue[i] <= 1 and gaugeValue[i] >= -1:
      gaugeSelect[i] = 0
      gaugeError[i] = 0
```

```
        else:
            if gaugeSelect[i] != 1:
                gaugeError[i] = 1
    return 0

def printAlertBox(x,y,alert,tag):
    # box
    if alert==0:
        pygame.draw.rect(screen,(233,150,122),arect(x,y,8,5))
    else:
        pygame.draw.rect(screen,redColor,arect(x,y,8,5))
    pygame.draw.rect(screen,blackColor,arect(x,y,8,5),3)

    # tag
    font = pygame.font.Font(None, 26)
    text = font.render(tag,1,whiteColor)
    screen.blit(text,arect(x+1.5,y+6,0,0))

def printGauge(x,y,val,tag,select):
    # horizontal bars
    pygame.draw.line(screen,whiteColor,(pw(x+2.2),ph(y)),(pw(x+3.8)
        ,ph(y)),5)
    pygame.draw.line(screen,whiteColor,(pw(x+2),ph(y+3)),(pw(x+4),
        ph(y+3)),5)
    pygame.draw.line(screen,redColor,(pw(x+2.2),ph(y+6)),(pw(x+3.8)
        ,ph(y+6)),5)
    pygame.draw.line(screen,whiteColor,(pw(x+2),ph(y+9)),(pw(x+4),
        ph(y+9)),5)
    pygame.draw.line(screen,redColor,(pw(x+2.2),ph(y+12)),(pw(x
        +3.8),ph(y+12)),5)
```

```
    pygame.draw.line(screen,whiteColor,(pw(x+2),ph(y+15)),(pw(x+4),
        ph(y+15)),5)
    pygame.draw.line(screen,whiteColor,(pw(x+2.2),ph(y+18)),(pw(x
        +3.8),ph(y+18)),5)

    # vertical bar
    pygame.draw.line(screen,whiteColor,(pw(x+3),ph(y)),(pw(x+3),ph(
        y+18)),5)

    # tag
    if select == 0:
      tagColor = blackColor
    else:
      tagColor = redColor
    pygame.draw.rect(screen,tagColor,arect(x+1,y+19,4,4))
    pygame.draw.rect(screen,blackColor,arect(x+1,y+19,4,4),3)
    font = pygame.font.Font(None, 26)
    text = font.render(tag,1,whiteColor)
    screen.blit(text,arect(x+2.5,y+19.8,0,0))



    # indicator
    pygame.draw.line(screen,whiteColor,(pw(x+0.5),ph(y+8+3*val)),(
        pw(x+1.8),ph(y+1+8+3*val)),4)
    pygame.draw.line(screen,whiteColor,(pw(x+0.5),ph(y+2+8+3*val))
        ,(pw(x+1.8),ph(y+1+8+3*val)),4)


# ++++++++++++++++++++++++++++++++++++++++++++++++
# ------------------------------------------------
```

```
# Panel 2: Tracking Task
# ----------------------------------------------------
# ++++++++++++++++++++++++++++++++++++++++++++++++++++

p2rect = arect(39,2,59,45)
uavError = [0,0,0]
boxList = []

#boxList.append(arect(60,20,5,5))

t1pos = (random.uniform(45,92),random.uniform(8,41))
t1dest = t1pos

t2pos = (random.uniform(45,92),random.uniform(8,41))
t2dest = t2pos

t3pos = (random.uniform(45,92),random.uniform(8,41))
t3dest = t3pos

def addBox(x,y):
  global boxList
  boxList.append(pygame.Rect(x-pw(2.5),y-ph(2.5),pw(5),ph(5)))

def updateBoxList():
  global boxList,t1pos,t2pos,t3pos,uavError
  uavError = [1,1,1]
  for b in boxList:
    if b.collidepoint(pw(t1pos[0]),ph(t1pos[1])):
      uavError[0] = 0
    if b.collidepoint(pw(t2pos[0]),ph(t2pos[1])):
```

```
        uavError[1] = 0
    if b.collidepoint(pw(t3pos[0]),ph(t3pos[1])):
        uavError[2] = 0
    if not b.collidepoint(pw(t1pos[0]),ph(t1pos[1])) and not b.
        collidepoint(pw(t2pos[0]),ph(t2pos[1])) and not b.
        collidepoint(pw(t3pos[0]),ph(t3pos[1])):
        boxList.remove(b)

def printBoxes():
    global boxList
    for b in boxList:
        pygame.draw.rect(screen,redColor,b,3)

def printTargets():
    global t1pos,t1dest,t2pos,t2dest,t3pos,t3dest

    pygame.draw.circle(screen,redColor,apr(t1pos),4)
    pygame.draw.circle(screen,greenColor,apr(t1dest),4)

    pygame.draw.circle(screen,redColor,apr(t2pos),4)
    pygame.draw.circle(screen,greenColor,apr(t2dest),4)

    pygame.draw.circle(screen,redColor,apr(t3pos),4)
    pygame.draw.circle(screen,greenColor,apr(t3dest),4)

def updateTargets(in_tock):
    global t1pos,t1dest,t2pos,t2dest,t3pos,t3dest,workloadLevel

    tock = in_tock * workloadLevel * 0.15
```

```
if t1pos == t1dest:
  t1dest = (random.uniform(45,92),random.uniform(8,41))
travel = (tock/1000.0)
dist = math.sqrt(math.pow(t1dest[1]-t1pos[1],2) + math.pow(
   t1dest[0]-t1pos[0],2))
if dist <= travel:
  t1pos=t1dest
else:
  theta = math.atan((t1dest[1]-t1pos[1])/(t1dest[0]-t1pos[0]))
  tx = travel*math.cos(theta)
  if t1dest[0] < t1pos[0]:
    tx = math.fabs(tx) * -1
  else:
    tx = math.fabs(tx)
  ty = travel*math.sin(theta)
  if t1dest[1] < t1pos[1]:
    ty = math.fabs(ty) * -1
  else:
    ty = math.fabs(ty)
  t1pos = (t1pos[0] + tx,t1pos[1] + ty)


if t2pos == t2dest:
  t2dest = (random.uniform(45,92),random.uniform(8,41))
dist = math.sqrt(math.pow(t2dest[1]-t2pos[1],2) + math.pow(
   t2dest[0]-t2pos[0],2))
if dist <= travel:
  t2pos=t2dest
else:
  theta = math.atan((t2dest[1]-t2pos[1])/(t2dest[0]-t2pos[0]))
  tx = travel*math.cos(theta)
```

```
  if t2dest[0] < t2pos[0]:
    tx = math.fabs(tx) * -1
  else:
    tx = math.fabs(tx)
  ty = travel*math.sin(theta)
  if t2dest[1] < t2pos[1]:
    ty = math.fabs(ty) * -1
  else:
    ty = math.fabs(ty)
  t2pos = (t2pos[0] + tx,t2pos[1] + ty)

if t3pos == t3dest:
  t3dest = (random.uniform(45,92),random.uniform(8,41))
dist = math.sqrt(math.pow(t3dest[1]-t3pos[1],2) + math.pow(
   t3dest[0]-t3pos[0],2))
if dist <= travel:
  t3pos=t3dest
else:
  theta = math.atan((t3dest[1]-t3pos[1])/(t3dest[0]-t3pos[0]))
  tx = travel*math.cos(theta)
  if t3dest[0] < t3pos[0]:
    tx = math.fabs(tx) * -1
  else:
    tx = math.fabs(tx)
  ty = travel*math.sin(theta)
  if t3dest[1] < t3pos[1]:
    ty = math.fabs(ty) * -1
  else:
    ty = math.fabs(ty)
  t3pos = (t3pos[0] + tx,t3pos[1] + ty)
```

```
# ++++++++++++++++++++++++++++++++++++++++++++++++
# ------------------------------------------------
# Panel 3: Assignment Task
# ------------------------------------------------
# ++++++++++++++++++++++++++++++++++++++++++++++++

p3rect = arect(2,52,35,45)

uav1 = []
uav2 = []
uav3 = []
selectedUAV = 0
currentTask = random.randint(1,5)
taskTime = 0
taskError = 1

def setPackages():
  global uav1,uav2,uav3
  uav1 = random.sample([1,2,3,4,5],random.randint(1,3))
  uav2 = random.sample([1,2,3,4,5],random.randint(1,3))
  uav3 = random.sample([1,2,3,4,5],random.randint(1,3))

  while list(set(uav1) | set(uav2) | set(uav3)) != [1,2,3,4,5]:
    missingCaps = list(set([1,2,3,4,5]) - (set(uav1) | set(uav2)
        | set(uav3)))
    randUAV = random.randint(1,3)
    if randUAV == 1:
      uav1 = list(set(uav1) | set(random.sample(missingCaps,1)))
    elif randUAV == 2:
```

```
        uav2 = list(set(uav2) | set(random.sample(missingCaps,1)))
    else:
        uav3 = list(set(uav3) | set(random.sample(missingCaps,1)))

setPackages()

def updateTasks(in_tock):
    global taskTime,currentTask,selectedUAV,workloadLevel,taskError

    tock = in_tock * workloadLevel * 0.75

    taskError = 1
    if selectedUAV == 1:
        if currentTask in uav1:
            taskError = 0
    elif selectedUAV == 2:
        if currentTask in uav2:
            taskError = 0
    elif selectedUAV == 3:
        if currentTask in uav3:
            taskError = 0


    taskTime = taskTime - tock
    if taskTime <= 0:
        selectedUAV = 0
        setPackages()
        currentTask = random.randint(1,5)
        taskTime = random.randint(7000,20000)
```

```
def printUAVBox(x,y,tag,tasks,select):
  # task boxes
  for task in tasks:
    #print x*task
    #print x*task+5
    pygame.draw.rect(screen,blueColor,arect(x+(task-1)*5,y+4,5,4)
        )

  # box
  if select == 0:
    backColor = greenColor
  else:
    backColor = redColor
  pygame.draw.rect(screen,backColor,arect(x,y,25,4))
  pygame.draw.rect(screen,blackColor,arect(x,y,25,8),3)

  # horizontal dividers
  pygame.draw.line(screen,blackColor,apr((x,y+4)),apr((x+24.8,y
      +4)),3)

  #vertical dividers
  pygame.draw.line(screen,blackColor,apr((x+5,y+4)),apr((x+5,y
      +7.8)),3)
  pygame.draw.line(screen,blackColor,apr((x+10,y+4)),apr((x+10,y
      +7.8)),3)
  pygame.draw.line(screen,blackColor,apr((x+15,y+4)),apr((x+15,y
      +7.8)),3)
  pygame.draw.line(screen,blackColor,apr((x+20,y+4)),apr((x+20,y
      +7.8)),3)
```

```
    # tag
    font = pygame.font.Font(None, 26)
    text = font.render(tag,1,whiteColor)
    screen.blit(text,arect(x+11,y+1,0,0))

    # task numbers
    font = pygame.font.Font(None, 26)
    text = font.render("1           2           3           4
        5",1,whiteColor)
    screen.blit(text,arect(x+2,y+5,0,0))

def printUAVTask(x,y,tag):
    # box
    pygame.draw.rect(screen,blueColor,arect(x,y,25,4))
    pygame.draw.rect(screen,blackColor,arect(x,y,25,4),3)

    # tag
    font = pygame.font.Font(None, 26)
    text = font.render(tag,1,whiteColor)
    screen.blit(text,arect(x+11,y+1,0,0))

#++++++++++++++++++++++++++++++++++++++++++++++++
#------------------------------------------------
# Panel 4: Resource Management Task
#------------------------------------------------
#++++++++++++++++++++++++++++++++++++++++++++++++

p4rect = arect(39,52,59,45)

tankALevel = 50
```

```
tankBLevel = 50
tankCLevel = 50
tankDLevel = 50

tankAError = 0
tankBError = 0

p1_val = 0
p2_val = 0
p3_val = 0
p4_val = 0
p5_val = 0
p6_val = 0
p7_val = 0
p8_val = 0

def updateTanks(in_tock):
  global tankALevel,tankBLevel,tankCLevel,tankDLevel
  global p1_val,p2_val,p3_val,p4_val,p5_val,p6_val,p7_val,p8_val
  global workloadLevel,tankAError,tankBError

  tock = in_tock * workloadLevel * 0.3

  tankALevel = tankALevel - tock*0.001
  tankBLevel = tankBLevel - tock*0.001

  tankAError = 0
  tankBError = 0

  if tankALevel > 60 or tankALevel < 40:
```

```
    tankAError = 1
  if tankBLevel > 60 or tankBLevel < 40:
    tankBError = 1

  if p1_val > 0:
    tankBLevel = tankBLevel + tock*0.002
    p1_val = p1_val - tock
    if p1_val == 0:
      p1_val = -1
  elif p1_val < 0:
    p1_val = p1_val - tock
    if p1_val <= -3000:
      p1_val = 0

  if p2_val > 0:
    if tankDLevel >= tock*0.001:
      tankBLevel = tankBLevel + tock*0.001
      tankDLevel = tankDLevel - tock*0.001
    p2_val = p2_val - tock
    if p2_val == 0:
      p2_val = -1
  elif p2_val < 0:
    p2_val = p2_val - tock
    if p2_val <= -3000:
      p2_val = 0

  if p3_val > 0:
    tankDLevel = tankDLevel + tock*0.001
    p3_val = p3_val - tock
    if p3_val == 0:
```

```
    p3_val = -1
elif p3_val < 0:
  p3_val = p3_val - tock
  if p3_val <= -3000:
    p3_val = 0

if p4_val > 0:
  tankALevel = tankALevel + tock*0.002
  p4_val = p4_val - tock
  if p4_val == 0:
    p4_val = -1
elif p4_val < 0:
  p4_val = p4_val - tock
  if p4_val <= -3000:
    p4_val = 0

if p5_val > 0:
  if tankCLevel >= tock*0.001:
    tankALevel = tankALevel + tock*0.001
    tankCLevel = tankCLevel - tock*0.001
  p5_val = p5_val - tock
  if p5_val == 0:
    p5_val = -1
elif p5_val < 0:
  p5_val = p5_val - tock
  if p5_val <= -3000:
    p5_val = 0

if p6_val > 0:
  tankCLevel = tankCLevel + tock*0.001
```

```
    p6_val = p6_val - tock
    if p6_val == 0:
      p6_val = -1
  elif p6_val < 0:
    p6_val = p6_val - tock
    if p6_val <= -3000:
      p6_val = 0

  if p7_val > 0:
    if tankBLevel >= tock*0.001:
      tankALevel = tankALevel + tock*0.001
      tankBLevel = tankBLevel - tock*0.001
    p7_val = p7_val - tock
    if p7_val == 0:
      p7_val = -1
  elif p7_val < 0:
    p7_val = p7_val - tock
    if p7_val <= -3000:
      p7_val = 0

  if p8_val > 0:
    if tankALevel >= tock*0.001:
      tankALevel = tankALevel - tock*0.001
      tankBLevel = tankBLevel + tock*0.001
    p8_val = p8_val - tock
    if p8_val == 0:
      p8_val = -1
  elif p8_val < 0:
    p8_val = p8_val - tock
    if p8_val <= -3000:
```

```
        p8_val = 0

    if tankALevel > 100:
      tankALevel = 100
    elif tankALevel < 0:
      tankALevel = 0

    if tankBLevel > 100:
      tankBLevel = 100
    elif tankBLevel < 0:
      tankBLevel = 0

    if tankCLevel > 100:
      tankCLevel = 100
    elif tankCLevel < 0:
      tankCLevel = 0

    if tankDLevel > 100:
      tankDLevel = 100
    elif tankDLevel < 0:
      tankDLevel = 0

def printTank(x,y,level,tag):
    # box
    #pygame.draw.rect(screen,blueColor,arect(x,y,16,19))
    level = 100 - level
    plevel = (level/100.0*19)
    pygame.draw.rect(screen,chargeColor,arect(x,y+plevel,16,19-
        plevel))
    pygame.draw.rect(screen,blackColor,arect(x,y,16,19),3)
```

```
    #level bound
    pygame.draw.line(screen,blackColor,apr((x+5,y+9.5)),apr((x+11,y
        +9.5)),5)
    pygame.draw.line(screen,redColor,apr((x+6,y+7.5)),apr((x+10,y
        +7.5)),3)
    pygame.draw.line(screen,redColor,apr((x+6,y+11.5)),apr((x+10,y
        +11.5)),3)
    pygame.draw.line(screen,blackColor,apr((x+7,y+5.5)),apr((x+9,y
        +5.5)),3)
    pygame.draw.line(screen,blackColor,apr((x+7,y+13.5)),apr((x+9,y
        +13.5)),3)
    #pygame.draw.line(screen,blackColor,apr((x-.2,y+7.5)),apr((x
        -.2,y+11.5)),5)
    #pygame.draw.line(screen,blackColor,apr((x+16.2,y+7.5)),apr((x
        +16.2,y+11.5)),5)

def printStorage(x,y,level,tag):
    # box
    #pygame.draw.rect(screen,blueColor,arect(x,y,16,19))
    level = 100 - level
    plevel = (level/100.0*6)
    pygame.draw.rect(screen,chargeColor,arect(x,y+plevel,5,6-plevel
        ))
    pygame.draw.rect(screen,blackColor,arect(x,y,5,6),3)

def printGenerator(x,y):
    pygame.draw.circle(screen,chargeColor,(pw(x),ph(y)),pw(2))
    pygame.draw.circle(screen,blackColor,(pw(x),ph(y)),pw(2),3)
```

```
def printPump(x,y,orientation,label,value):
  if value == 0:
    pumpColor = whiteColor
  elif value < 0:
    pumpColor = redColor
  elif value < 5000:
    pumpColor = yellowColor
  else:
    pumpColor = greenColor

  if orientation == "UP":
    pygame.gfxdraw.filled_trigon(screen, pw(x-1), ph(y+1), pw(x
        +1), ph(y+1), pw(x), ph(y-1), blackColor)
    # label
    font = pygame.font.Font(None, 22)
    text = font.render(label,1,whiteColor)
    screen.blit(text,arect(x+1.3,y-0.3,0,0))
    pygame.draw.circle(screen,pumpColor,apr((x,y+0.3)),pw(0.3))
  elif orientation == "DOWN":
    pygame.gfxdraw.filled_trigon(screen, pw(x-1), ph(y-1), pw(x
        +1), ph(y-1), pw(x), ph(y+1), blackColor)
    # label
    font = pygame.font.Font(None, 26)
    text = font.render(label,1,whiteColor)
    screen.blit(text,arect(x+2.5,y+1,0,0))
    pygame.draw.circle(screen,pumpColor,apr((x,y+0.3)),pw(0.3))
  elif orientation == "LEFT":
    pygame.gfxdraw.filled_trigon(screen, pw(x+0.7), ph(y-1.3), pw
        (x+0.7), ph(y+1.3), pw(x-0.7), ph(y), blackColor)
    # label
```

```
        font = pygame.font.Font(None, 22)
        text = font.render(label,1,whiteColor)
        screen.blit(text,arect(x-0.3,y+1.9,0,0))
        pygame.draw.circle(screen,pumpColor,apr((x+0.3,y)),pw(0.3))
    elif orientation == "RIGHT":
        pygame.gfxdraw.filled_trigon(screen, pw(x-0.7), ph(y-1.3), pw
            (x-0.7), ph(y+1.3), pw(x+0.7), ph(y), blackColor)
        # label
        font = pygame.font.Font(None, 22)
        text = font.render(label,1,whiteColor)
        screen.blit(text,arect(x-0.3,y-2.9,0,0))
        pygame.draw.circle(screen,pumpColor,apr((x-0.3,y)),pw(0.3))

def printPipes():
    pygame.draw.line(screen,blackColor,apr((50,77)),apr((50,85)),3)
    pygame.draw.line(screen,blackColor,apr((58,77)),apr((58,85)),3)

    pygame.draw.line(screen,blackColor,apr((81,77)),apr((81,85)),3)
    pygame.draw.line(screen,blackColor,apr((89,77)),apr((89,85)),3)

    pygame.draw.line(screen,blackColor,apr((51,87)),apr((57,87)),3)
    pygame.draw.line(screen,blackColor,apr((82,87)),apr((88,87)),3)

    pygame.draw.line(screen,blackColor,apr((61,62)),apr((76,62)),3)
    pygame.draw.line(screen,blackColor,apr((61,73)),apr((76,73)),3)




# ++++++++++++++++++++++++++++++++++++++++++++++++
# ------------------------------------------------
```

```
# Main Program Code
# ------------------------------------------------------
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++

# p1rect, p2rect, p3rect, p4rect

#panel1 = pygame.Rect(x-pw(2.5),y-ph(2.5),pw(5),ph(5))

# panel 1 bounds
alert1_r = arect(4,4,12,11)
alert2_r = arect(21,4,12,11)
inda_r = arect(4,18,6,26)
indb_r = arect(12,18,6,26)
indc_r = arect(20,18,6,26)
indd_r = arect(28,18,6,26)

# panel 3 bounds
uav1_r = arect(6,55,27,10)
uav2_r = arect(6,66,27,10)
uav3_r = arect(6,77,27,10)

# panel 4 bounds
p1_r = arect(87,79,4,4)
p2_r = arect(79,79,4,4)
p3_r = arect(82.5,85,4,4)
p4_r = arect(56,79,4,4)
p5_r = arect(48,79,4,4)
p6_r = arect(51.5,85,4,4)
p7_r = arect(66.5,71,4,4)
p8_r = arect(66.5,60,4,4)
```

```
def printInRects():
    pygame.draw.rect(screen,pinkColor,alert1_r,2)
    pygame.draw.rect(screen,pinkColor,alert2_r,2)
    pygame.draw.rect(screen,pinkColor,inda_r,2)
    pygame.draw.rect(screen,pinkColor,indb_r,2)
    pygame.draw.rect(screen,pinkColor,indc_r,2)
    pygame.draw.rect(screen,pinkColor,indd_r,2)

    pygame.draw.rect(screen,pinkColor,uav1_r,2)
    pygame.draw.rect(screen,pinkColor,uav2_r,2)
    pygame.draw.rect(screen,pinkColor,uav3_r,2)

    pygame.draw.rect(screen,pinkColor,p1_r,2)
    pygame.draw.rect(screen,pinkColor,p2_r,2)
    pygame.draw.rect(screen,pinkColor,p3_r,2)
    pygame.draw.rect(screen,pinkColor,p4_r,2)
    pygame.draw.rect(screen,pinkColor,p5_r,2)
    pygame.draw.rect(screen,pinkColor,p6_r,2)
    pygame.draw.rect(screen,pinkColor,p7_r,2)
    pygame.draw.rect(screen,pinkColor,p8_r,2)

def input(events):
    global simState,workloadLevel,cycleCountDown

    # Panel 1 globals
    global gaugeSelect,alertVal

    # Panel 3 globals
    global selectedUAV,uav1_r,uav2_r,uav3_r
```

```
# Panel 4 globals
global p1_val, p2_val, p3_val, p4_val, p5_val, p6_val, p7_val, p8_val

for event in events:
  if event.type == QUIT:
    sys.exit(0)
  elif event.type == KEYDOWN:
    if event.key == K_ESCAPE:
      pygame.event.post(pygame.event.Event(QUIT))
  elif event.type == MOUSEBUTTONUP:
    if simState == 0:
      if event.button == 3:
        resetSim()
        simState = 1
        workloadLevel = workloadLevel + 1
        cycleCountDown = trialTime
    elif simState == 1:
      if event.button == 1:
        #print "(" + str(event.pos[0]) + "," + str(event.pos
            [1]) + ")  (" + str(p1rect[0]) + "," + str(p1rect
            [1]) + "," + str(p1rect[2]) + "," + str(p1rect[3]) +
             ")"
        if p1rect.collidepoint(event.pos[0],event.pos[1]):
          # Panel 1 Inputs
          if alert1_r.collidepoint(event.pos[0],event.pos[1]):
            alertVal[0] = 0
          elif alert2_r.collidepoint(event.pos[0],event.pos[1])
              :
            alertVal[1] = 0
```

```
        elif inda_r.collidepoint(event.pos[0],event.pos[1]):
            gaugeSelect[0] = 1
        elif indb_r.collidepoint(event.pos[0],event.pos[1]):
            gaugeSelect[1] = 1
        elif indc_r.collidepoint(event.pos[0],event.pos[1]):
            gaugeSelect[2] = 1
        elif indd_r.collidepoint(event.pos[0],event.pos[1]):
            gaugeSelect[3] = 1


    elif p2rect.collidepoint(event.pos[0],event.pos[1]):
        # Panel 2 Inputs
        addBox(event.pos[0],event.pos[1])

    elif p3rect.collidepoint(event.pos[0],event.pos[1]):
        # Panel 3 Inputs
        if uav1_r.collidepoint(event.pos[0],event.pos[1]):
            selectedUAV = 1
        elif uav2_r.collidepoint(event.pos[0],event.pos[1]):
            selectedUAV = 2
        elif uav3_r.collidepoint(event.pos[0],event.pos[1]):
            selectedUAV = 3


    elif p4rect.collidepoint(event.pos[0],event.pos[1]):
        # Panel 4 Inputs
        if p1_r.collidepoint(event.pos[0],event.pos[1]):
            if p1_val > 0:
                p1_val = 0
            elif p1_val == 0:
```

```
            p1_val = 15000
        elif p2_r.collidepoint(event.pos[0],event.pos[1]):
          if p2_val > 0:
            p2_val = 0
          elif p2_val == 0:
            p2_val = 15000
        elif p3_r.collidepoint(event.pos[0],event.pos[1]):
          if p3_val > 0:
            p3_val = 0
          elif p3_val == 0:
            p3_val = 15000
        elif p4_r.collidepoint(event.pos[0],event.pos[1]):
          if p4_val > 0:
            p4_val = 0
          elif p4_val == 0:
            p4_val = 15000
        elif p5_r.collidepoint(event.pos[0],event.pos[1]):
          if p5_val > 0:
            p5_val = 0
          elif p5_val == 0:
            p5_val = 15000
        elif p6_r.collidepoint(event.pos[0],event.pos[1]):
          if p6_val > 0:
            p6_val = 0
          elif p6_val == 0:
            p6_val = 15000
        elif p7_r.collidepoint(event.pos[0],event.pos[1]):
          if p7_val > 0:
            p7_val = 0
          elif p7_val == 0:
```

```
            p7_val = 15000
        elif p8_r.collidepoint(event.pos[0],event.pos[1]):
          if p8_val > 0:
            p8_val = 0
          elif p8_val == 0:
            p8_val = 15000


def resetSim():
  random.seed(simSeed)
  resetPanel1()
  resetPanel2()
  resetPanel3()
  resetPanel4()

def resetPanel1():
  global gaugeValue,gaugeTarget,gaugeSelect,alertTime,alertVal
  gaugeValue = [0,0,0,0]
  gaugeTarget = [0,0,0,0]
  gaugeSelect = [0,0,0,0]

  alertTime = [random.randint(10000,30000),random.randint
     (10000,30000)]
  alertVal = [0,0]

def resetPanel2():
  global boxList,t1pos,t1dest,t2pos,t2dest,t3pos,t3dest
  boxList = []

  t1pos = (random.uniform(45,92),random.uniform(8,41))
```

```
    t1dest = t1pos

    t2pos = (random.uniform(45,92),random.uniform(8,41))
    t2dest = t2pos

    t3pos = (random.uniform(45,92),random.uniform(8,41))
    t3dest = t3pos

def resetPanel3():
    global uav1,uav2,uav3,selectedUAV,currentTask,taskTime
    uav1 = []
    uav2 = []
    uav3 = []
    selectedUAV = 0
    currentTask = random.randint(1,5)
    taskTime = 0

def resetPanel4():
    global p1_val,p2_val,p3_val,p4_val,p5_val,p6_val,p7_val,p8_val
    global tankALevel,tankBLevel,tankCLevel,tankDLevel
    tankALevel = 50
    tankBLevel = 50
    tankCLevel = 50
    tankDLevel = 50

    p1_val = 0
    p2_val = 0
    p3_val = 0
    p4_val = 0
    p5_val = 0
```

```
    p6_val = 0
    p7_val = 0
    p8_val = 0


#ser = serial.Serial(port='COM3', baudrate=19200)

fileName = "Subject_" + str(sys.argv[1]) + "_Results.txt"

outFile = open(fileName,'w')
# time, workload, gsr, alert1, alert2, ind_a, ind_b, ind_c, ind_d
    , track_1, track_2, track_3, task_cover, tank_A, tank_B
#outFile.write("this is a test\n")

startTime = time.time()
simTime = 0
tockTime = 0

gsrVal = 0

while True:
    # gsrVal = 0
    input(pygame.event.get())

    newsimTime = long((time.time()-startTime)*1000)
    tockTime = newsimTime - simTime
    simTime = newsimTime

# s = ser.read(ser.inWaiting())
# if len(s)>0:
```

```
#    #print str(s[0],encoding='utf8')
#    #print ord(s[0])
#    gsrVal = ord(s[0])

    if cycleCountDown <= 0:
      cycleCountDown = 0
      if workloadLevel == 10:
        simState = 2
      else:
        simState = 0

    if simState == 0:
      # Display start screen
      screen.fill((255,248,220))

      font = pygame.font.Font(None, 100)
      text = font.render("Right-Click to Begin " + str(
          workloadLevel+1) + "/10",1,blackColor)
      screen.blit(text,arect(17,45,0,0))

    elif simState == 2:
      screen.fill((255,248,220))

      font = pygame.font.Font(None, 100)
      text = font.render("Experiment Complete",1,blackColor)
      screen.blit(text,arect(22,45,0,0))

    elif simState == 1:
```

```
outFile.write(str(simTime) + ',' + str(workloadLevel) + ',' +
    str(gsrVal) + ',' + \
  str(alertVal[0]) + ',' + str(alertVal[1]) + ',' + \
  str(gaugeError[0]) + ',' + str(gaugeError[1]) + ',' + str(
    gaugeError[2]) + ',' + str(gaugeError[3]) + ',' + \
  str(uavError[0]) + ',' + str(uavError[1]) + ',' + str(
    uavError[2]) + ',' + \
  str(taskError) + ',' + \
  str(tankAError) + ',' + str(tankBError) + '\n')

cycleCountDown = cycleCountDown - tockTime

#print tockTime



screen.fill((255,248,220))

# Panel 1
pygame.draw.rect(screen,frameColor,p1rect)
pygame.draw.rect(screen,borderColor,p1rect,4)

updateGauges(tockTime)
printGauge(4,20,gaugeValue[0],"A",gaugeSelect[0])
printGauge(12,20,gaugeValue[1],"B",gaugeSelect[1])
printGauge(20,20,gaugeValue[2],"C",gaugeSelect[2])
printGauge(28,20,gaugeValue[3],"D",gaugeSelect[3])

updateAlerts(tockTime)
printAlertBox(6,6,alertVal[0],"ALERT 1")
printAlertBox(23,6,alertVal[1],"ALERT 2")
```

```
# Panel 2
pygame.draw.rect(screen,frameColor,p2rect)
pygame.draw.rect(screen,borderColor,p2rect,4)

updateTargets(tockTime)
printTargets()

updateBoxList()
printBoxes()

# Panel 3
updateTasks(tockTime)

pygame.draw.rect(screen,frameColor,p3rect)
pygame.draw.rect(screen,borderColor,p3rect,4)

uavList = [0,0,0,0]
uavList[selectedUAV] = 1
printUAVBox(7,56,"UAV 1",uav1,uavList[1])
printUAVBox(7,67,"UAV 2",uav2,uavList[2])
printUAVBox(7,78,"UAV 3",uav3,uavList[3])

printUAVTask(7,89,"Task " + str(currentTask))

# Panel 4
pygame.draw.rect(screen,frameColor,p4rect)
pygame.draw.rect(screen,borderColor,p4rect,4)

updateTanks(tockTime)
```

```
    printPipes()

    printTank(45,58,tankALevel,"A")
    printTank(76,58,tankBLevel,"B")

    printStorage(46,85,tankCLevel,"C")
    printStorage(77,85,tankDLevel,"D")

    printGenerator(58,88)
    printGenerator(89,88)

    printPump(89,81,"UP","P1",p1_val)
    printPump(81,81,"UP","P2",p2_val)
    printPump(84.5,87,"LEFT","P3",p3_val)

    printPump(58,81,"UP","P4",p4_val)
    printPump(50,81,"UP","P5",p5_val)
    printPump(53.5,87,"LEFT","P6",p6_val)

    printPump(68.5,73,"LEFT","P7",p7_val)
    printPump(68.5,62,"RIGHT","P8",p8_val)

    # Input Boxes
    #printInRects()

# ------------------------------------------------
# ---------- Other Display Stuff ----------------
# ------------------------------------------------
```

```
pygame.display.update()
fpsClock.tick(60)
```

Bibliography

1. AFRL. Wpafb afrl media gallery, FEB 2012. http://www.wpafb.af.mil/art/mediagallery.asp?galleryID=2633.

2. Arnegard, R., and Comstock, J. Multi-attribute task battery-applications in pilot workload and strategic behavior research. In *International Symposium on Aviation Psychology 6th* (1991), pp. 1118–1123.

3. Bennett, K., Cress, J., Hettinger, L., Stautberg, D., and Haas, M. A theoretical analysis and preliminary investigation of dynamically adaptive interfaces. *The International Journal of Aviation Psychology 11*, 2 (2001), 169–195.

4. Bungie Studios. Halo: Combat evolved. Xbox, 2000.

5. Bunt, A., Conati, C., and McGrenere, J. What role can adaptive support play in an adaptable system? In *Proceedings of the 9th International Conference on Intelligent User Interfaces* (New York, NY, USA, 2004), IUI '04, ACM, pp. 117–124.

6. Bunt, A., Conati, C., and McGrenere, J. Supporting interface customization using a mixed-initiative approach. In *Proceedings of the 12th International Conference on Intelligent User Interfaces* (New York, NY, USA, 2007), IUI '07, ACM, pp. 92–101.

7. Butler, R., Miller, S., Potts, J., and Carreno, V. A formal methods approach to the analysis of mode confusion. In *Digital Avionics Systems Conference, 1998. Proceedings., 17th DASC. The AIAA/IEEE/SAE* (1998), vol. 1, IEEE, pp. C41–1.

8. Calhoun, G., Ruff, H., Draper, M., and Wright, E. Automation-level transference effects in simulated multiple unmanned aerial vehicle control. *Journal of Cognitive Engineering and Decision Making 5*, 1 (2011), 55.

9. Calhoun, G., Ward, V., and Ruff, H. Performance-based adaptive automation for supervisory control. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* (2011), vol. 55, SAGE Publications, pp. 2059–2063.

10. Dixon, S., and Wickens, C. Automation reliability in unmanned aerial vehicle control: A reliance-compliance model of automation dependence in high workload. *Human Factors: The Journal of the Human Factors and Ergonomics Society 48*, 3 (2006), 474–486.

11. Electronic Arts. Battlefield 1942. CD-ROM, 2002.

12. Endsley, M. Level of automation effects on performance, situation awareness and workload in a dynamic control task. *Ergonomics 42*, 3 (1999), 462–492.

13. Findlater, L., and McGrenere, J. A comparison of static, adaptive, and adaptable menus. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2004), ACM, pp. 89–96.

14. Hackos, J., and Redish, J. *User and Task Analysis for Interface Design*. Wiley New York, 1998.

15. Hk, K. Steps to take before intelligent user interfaces become real. *Interacting with Computers 12*, 4 (2000), 409 – 426.

16. Horvitz, E. Principles of mixed-initiative user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: the CHI is the Limit* (New York, NY, USA, 1999), CHI '99, ACM, pp. 159–166.

17. Hou, M., Zhu, H., Zhou, M., and Arrabito, G. Optimizing operator-agent interaction in intelligent adaptive interface design: A conceptual framework. *IEEE Transactions on Systems, Man, and Cybernetics, Part C* (2011), 161–178.

18. Jameson, A. Adaptive interfaces and agents. In *Human-Computer Interaction Handbook*, J. A. Jacko and A. Sears, Eds. Erlbaum, Mahwah, NJ, 2003, pp. 305–330. Available from http://dfki.de/∼jameson/abs/Jameson03Handbook.html.

19. Kaber, D., Perry, C., Segall, N., McClernon, C., and Prinzel, L. Situation awareness implications of adaptive automation for information processing in an air traffic control-related task. *International Journal of Industrial Ergonomics 36*, 5 (2006), 447–462.

20. Kawamura, K., Nilas, P., Muguruma, K., Adams, J., and Zhou, C. An agent-based architecture for an adaptive human-robot interface. In *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on* (2003), IEEE, pp. 8–pp.

21. Mekdeci, B., and Cummings, M. Modeling multiple human operators in the supervisory control of heterogeneous unmanned vehicles. In *Proceedings of the 9th Workshop on Performance Metrics for Intelligent Systems* (New York, NY, USA, 2009), PerMIS '09, ACM, pp. 1–8.

22. Miller, G. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review 63*, 2 (1956), 81.

23. Mo, J., La, R., Anantharam, V., and Walrand, J. Analysis and comparison of tcp reno and vegas. In *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE* (1999), vol. 3, IEEE, pp. 1556–1563.

24. Nachreiner, F. Standards for ergonomics principles relating to the design of work systems and to mental workload. *Applied Ergonomics 26*, 4 (1995), 259–263.

25. NASA. Nasa tlx: Task load index, dec 2011. http://human-factors.arc.nasa.gov/groups/TLX/.

26. NASA. Official matb website :: Nasa langley research center, jan 2012. http://matb.larc.nasa.gov.

27. Parasuraman, R. Theory and design of adaptive automation in aviation systems. Tech. rep., DTIC Document, 1992.

28. Parasuraman, R., Sheridan, T., and Wickens, C. Situation awareness, mental workload, and trust in automation: Viable, empirically supported cognitive engineering constructs. *Journal of Cognitive Engineering and Decision Making 2*, 2 (2008), 140.

29. Parasuraman, R., and Wickens, C. Humans: Still vital after all these years of automation. *Human Factors: The Journal of the Human Factors and Ergonomics Society 50*, 3 (2008), 511–520.

30. Pisano, P., Pol, J., Goodwin, L., and Stern, A. Fhwas clarus initiative: Concept of operations and associated research. In *Proceedings of the 22nd Conference on Interactive Information and Processing Systems* (2005).

31. Prinzel, L., Freeman, F., Scerbo, M., Mikulka, P., and Pope, A. A closed-loop system for examining psychophysiological measures for adaptive task allocation. *The International Journal of Aviation Psychology 10*, 4 (2000), 393–410.

32. Pygame Developer Community. Pygame - python game development, Jan 2012. http://pygame.org/.

33. Python Software Foundation. Python programming language - official website, Jan 2012. http://python.org/.

34. Ruff, H., and Calhoun, G. Impact of automation level and reliability on multi-vehicle supervisory control task performance. In *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit* (2011), Infotech(at)Aerospace.

35. Russell, S., Norvig, P., Candy, J., Malik, J., and Edwards, D. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

36. Salvendy, G. *Handbook of Human Factors and Ergonomics*. John Wiley & Sons New York, NY, 1997.

37. Scallen, S., and Hancock, P. Implementing adaptive function allocation. *The International Journal of Aviation Psychology 11*, 2 (2001), 197–221.

38. Scallen, S., Hancock, P., and Duley, J. Pilot performance and preference for short cycles of automation in adaptive function allocation. *Applied Ergonomics 26*, 6 (1995), 397–403.

39. Sheridan, T., and Parasuraman, R. Human-automation interaction. *Reviews of human factors and ergonomics 1*, 1 (2005), 89.

40. Shi, Y., Choi, E., Ruiz, N., Chen, F., and Taib, R. Galvanic skin response (gsr) as an index of cognitive workload. In *ACM CHI Conference Work-in-Progress* (2007).

41. Smith, K. Wickens' multiple resource theory (mrt) model, FEB 2012. http://en.wikipedia.org/wiki/File:KTS1workload.jpg.

42. Sony Online Entertainment. Planetside. DVD-ROM, 2003.

43. Sutton, R., and Barto, A. *Introduction to Reinforcement Learning*. MIT Press, 1998.

44. Tejada, S., Tarapore, S., Cristina, A., Goodwyne, P., and O'Hara, R. Mixed-initiative interface for human, robot, agent collaboration in urban search and

rescue teams. In *Automation Congress, 2004. Proceedings. World* (2004), vol. 15, IEEE, pp. 467–472.

45. Tulga, M., and Sheridan, T. Dynamic decisions and work load in multitask supervisory control. *Systems, Man and Cybernetics, IEEE Transactions on 10*, 5 (1980), 217–232.

46. Whitlow, S., Dorneich, M., Funk, H., and Miller, C. Providing appropriate situation awareness within a mixed-initiative control system. In *Systems, Man and Cybernetics, 2002 IEEE International Conference on* (2002), vol. 5, IEEE, pp. 5–pp.

47. Wickens, C. Processing resources and attention. *Multiple-Task Performance* (1991), 3–34.

48. Wilson, G., Lambert, J., and Russell, C. Performance enhancement with real-time physiologically controlled adaptive aiding. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* (2000), vol. 44, SAGE Publications, pp. 61–64.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
22-03-2012	Master's Thesis	Aug 2010 - Mar 2012

4. TITLE AND SUBTITLE
Workload-Based Automated Interface Mode Selection

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Compton, Andrew J M, Capt

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
Wright-Patterson AFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCE/ENG/12-03

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Intentionally Left Blank

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
DISTRIBUTION A. Approved for public release: distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The increase in the size of the Air Force's Unmanned Aerial Vehicle (UAV) fleet, and the desire to reduce operational manning requirements, has led to an interest in Multiple Aircraft Control (MAC) technology. The MAC concept is highly prone to operator overload, as it requires operators to maintain awareness for multiple aircraft. To attempt to mitigate the potential of operator overload, this research introduces an agent into the system interface to assume responsibility for managing automation mode selection. The agent uses a novel dynamic scheme for determining how and when to introduce automation assistance to the operator. By using a reinforcement learning approach, the interface agent is able to correlate an operator's workload and performance levels. This allows the agent to determine the most appropriate times to introduce automation assistance. By automating tasks at appropriate times, the agent helps the system balance the operator's workload level, striking the best possible balance between operator awareness and overall performance, while reducing the potential for operator overload.

15. SUBJECT TERMS
Interface Automation, Multiple Aircraft Control, Uninhabited Aerial Vehicle, Automation Mode Selection, Operator Overload, Automation Overreliance, Automation Brittleness, Workload

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Gilbert L. Peterson
U	U	U	UU	120	19b. TELEPHONE NUMBER *(Include area code)* (937) 255-6565 x4281 Gilbert.Peterson@afit.edu